An Approach to Optimize Intra-ECU Communication Based on Mapping of AUTOSAR Runnable Entities

Rongshen Long, Hong Li , Wei Peng, Yi Zhang, Minde Zhao
College of Computer Science, Zhejiang University
Hangzhou 310027, Zhejiang, China
{samlrs, lihong, xiao4, stone_cold, zmdd48}@zju.edu.cn

Abstract

Communication plays an important role in modern automotive system architecture. In order to optimize the AUTOSAR intra-ECU communication, an approach based on mapping runnable entities is proposed to reduce task switching, and avoid communication data inconsistency and time delay. Taking into account the various communication behaviors of all runnable entities in the current ECU, we provide six extended rules based on AUTOSAR RTE mapping rules to properly group the runnables into the OS tasks. In addition, an equation is given to evaluate the mapping performance. A mapping case of a vehicle control application has been successfully implemented on SmartOSEK OS.

1. Introduction

In recent years, contemporary automotive E/E architecture has reached a high level of complexity which calls for a technological breakthrough in order to satisfy the growing requirements of innovative vehicle applications. This need is particularly notable for high-end, luxury vehicle manufacturers.

AUTOSAR [1] (Automotive Open System ARchi-tecture) solution meets the challenge. AUTOSAR is an open industry standard for automotive E/E architectures which will serve as a basic infrastructure for the management of functions within both future applications and standard software modules. Initially held by BMW, Bosch, Continental, Daimler- Chrysler and Volkswagen in August, 2002, this advanced architecture has acquired 10 core partners, 52 premium members and 45 associate members by 2006. Now AUTOSAR reaches its 3.1 version and becomes a reliable international automotive E/E standard.

SmartSAR is a software platform for automotive electronics developed by the Embedded System and Engineering Center in Zhejiang University. Following the international AUTOSAR specification, SmartSAR platform makes a clear layered view of automotive electronics software, which separates application software and the underlying software completely by means of loosing coupling mechanism of components and software run-time environment.

According to AUTOSAR, the run-time environment of SmartSAR is responsible for mapping the components' runnable entities to SmartOSEK OS [2] (the SmartSAR OS) tasks. However, AUTOSAR only provides a few basic rules for mapping runnable entities. In SmartSAR, we extend and improve the AUTOSAR mapping rules in order to optimize the intra-ECU (Electronic Control Unit) communication.

This paper is organized as follows. In the next section, we introduce basic concepts and the architecture of AUTOSAR. Then we present the key issues of intra-ECU communication in section 3. In section 4, we analyze the extended mapping rules and give an evaluation equation to evaluate the mapping performance. Section 5 presents a case-study, in which the runnable entities of a vehicle application were mapped into tasks in SmartOSEK OS. Finally, the concluding section summarizes this paper and touches on possible future work.

2. Basic concepts and definitions

2.1. The AUTOSAR architecture

AUTOSAR aims to fulfill the future vehicle requirements, such as availability and safety, software upgrades/updates and maintainability. It gives the definition of a modular software architecture and standardization of different APIs to separate the AUTOSAR software layers and provide high modularity and configurability. Figure 1 show its layered software architecture.

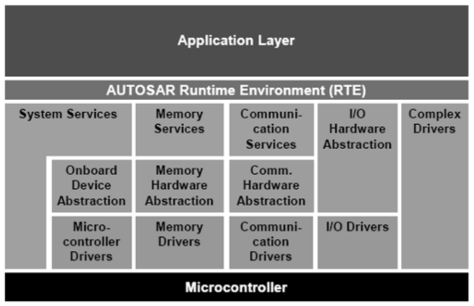

Figure 1. AUTOSAR Software Architecture [1].

To support the integration of E/E architecture, AUTOSAR brings in a conceptual communication mechanism called Virtual Functional Bus (VFB). In AUTOSAR, an application is modeled as a composition of interconnected components. VFB allows for a strict separation between applications and infrastructure. The software components are quite independent of the concrete communication mechanisms through which one component interacts with other components or with hardware.

AUTOSAR also provides a methodology and standardized workflows for development. A common technical approach is called the "AUTOSAR Methodology" which describes all major steps of the development of a system with AUTOSAR: from the system-level configuration to the generation of ECU executable codes.

2.2. AUTOSAR component and runnable entity

In AUTOSAR, "application" software is conceptually located above the AUTOSAR RTE and consists of "AUTOSAR application software-components" that are ECU and location independent [3]. A component has several "ports", through which ones component can interact with other components. A port of a component is either a "PPort" or an "RPort". A "PPort" provides the elements while an "RPort" requires the elements. A port of a component is associated with a "port-interface". Two types of port-interfaces, including client-server, sender-receiver, are supported by AUTOSAR. A client-server interface defines a set of operations that can be invoked by a client and implemented by a server. A sender-receiver interface defines a set of data-elements that are sent and received over the VFB.

Runnable Entities (runnable for short) are the smallest code fragments in AUTOSAR software-components. One component consists of at least one runnable. All runnables are activated by RTEEvents. If none RTEEvent is specified as StartOnEvent for the runnable, the runnable is never activated by the RTE. In scope of the RTE, two categories of runnables are used. Category 1 runnables do not have WaitPoints and have to terminate in finite time. runnables of category 2 always have at least one WaitPoint or they invoke a server and wait for the response to return [4].

At the ECU level, AUTOSAR defines software components as atomic entities from a software development view. However, when it comes to scheduling, runnables belonging to different software components are then grouped into tasks, which are finally put under operating system control [5]. The RTE generator is responsible for constructing the OS tasks bodies. Before the RTE generator takes the ECU configuration description as the input information to generate the codes, the RTE configurator configures parts of the ECU-Configuration, e.g. the mapping of runnables to tasks, which is the main focus in this paper.

2.3. AUTOSAR OS task and RTEEvent

Runnable is executed in the context of an OS task. Two different types of task are provided by the AUTOSAR OS [6]: basic task and extended task. Extended tasks are distinguished from basic tasks by being allowed to call WaitEvent, which may result in a waiting state [7].

Runnables are triggered by RTEEvents. AUTOSAR RTE supports seven types of RTEEvents, they are TimingEvent, DataReceivedEvent, DataReceive-ErrorEvent, DataSendCompletedEvent, OperationIn-vokedEvent AsynchronousServerCallReturnsEvent and ModeSwitchEvent. They are generally implemented by AUTOSAR OS event, except TimingEvent and ModeSwitchEvent. TimingEvent can be implemented by OS alarm.

2.4. AUTOSAR RTE

As can be seen from Figure 1, RTE is located in the midst of the application layer and the AUTOSAR infrastructure layers. It looks like a functional bus to connect AUTOSAR components. In fact, the RTE is at the heart of the AUTOSAR ECU architecture, and it is the realization (for a particular ECU) of the interfaces of the AUTOSAR VFB [4]. RTE maps all runnables of each component in local ECU to the OS tasks and builds the existing intra-ECU communications among them. If necessary, RTE is also responsible for building inter-ECU communications between several runnables located in different ECUs.

RTE's communication models have three patterns: Sender-Receiver, Client-Server and InterRunnable-Variables. Sender-Receiver and Client-Server are used for communications between AUTOSAR software-componets, while InterRunnableVariables are established between runnables inside one AUTOSAR software component. Sender-receiver communication involves the transmission and reception of signals consisting of atomic data elements that are sent by one runnable and received by one or more runnables. Client-server communication involves two entities, the client which is the demander of a service and the server that provides the service. InterRunnableVariables can only be accessed by runnables of one AUTOSAR software-component. These runnables might be running in the same or in different task contexts.

Each runnable should include at least one communication behavior of the following: DataReadAccess, DataWriteAccess, ServerCallPoint, DataSendPoint, DataReceivePoint or WaitPoint. In AUTOSAR, Runnables using DataReadAccess or DataWriteAccess belong to category 1. Runnables in category 2 do not allow DataReadAccess or DataWriteAccess.

2.5. The AUTOSAR rules of mapping runnable entities to tasks

Mapping runnables to build OS tasks bodies is one of the main requirements of RTE. Essentially the mapping depends on different communication behaviors of runnables mentioned above.

In RTE specification, three rules suggested by AUTOSAR should be taken into account [4]:

B.1) *Runnable entities of category 1 can be mapped either to basic or extended tasks.*

B.2) *Runnables of category 2 that contain WaitPoints will be typically mapped to extended tasks.*

B.3) *Runnables of category 2 that contain a SynchronousServerCallPoint generally have to be mapped to extended tasks. If no timeout monitoring is required, or if the server runnable is invoked directly and is itself of category 1, they can be mapped to basic tasks.*

We consider these rules as AUTOSAR basic rules for mapping runnables in this paper.

3. Key issues of intra-ECU communication

3.1. Comparison of the basic and extended task

After being mapped to ECUs, all application software components (exactly all runnables) in the same ECU will run in OS tasks and interact with each other via intra-ECU communications.

So the first issue, which should be considered, is to choose a basic or extended task to map the runnables. Basic tasks of AUTOSAR OS (OSEK OS) have no waiting state, and thus only comprise synchronization points at the beginning and at the end of the task. An advantage of basic tasks is their moderate requirements regarding run time context (RAM) [6]. Extended tasks comprise more synchronization points than basic tasks.

In addition, in order to have a deterministic execution timing, it’s better to choose a basic task to map the runnable, if we could.

3.2. Frequent task-switching

Intra-ECU communication includes both intra-task and inter-task distribution. Intra-task communication occurs between runnables that are mapped to the same OS task whereas inter-task communication is for runnables mapped to different tasks and therefore it can involve a context switch. Generally, the data-elements delivered through intra-ECU communications are implemented by global variables.

In intra-ECU communication, frequent task switching may occur if we map the runnables improperly. RTEEvent is used to activate a runnable or wake it up from its WaitPoint. Several operations called by runnables will trigger RTEEvents. For example, when a runnable calls Rte_Send API to send a data, a DataReceivedEvent will be triggered automatically when the data reaches its destination.

Let’s take the case with one sender and multiple receivers in intra-ECU communication as example. If we map each receiver into different tasks in this case, which is considered as improper mapping, task switching will occur frequently because RTE will activate (or wake up) each receiver’s task one by one after the sender sends its data. This will make the intra-ECU communication inefficient. In this paper, we will propose a mapping rule to avoid this inefficiency.

3.3. Data consistency

Obviously, concurrent accesses to shared data memory can lead to data inconsistency. However, intra-ECU communications generally bring in global variables which will be accessed by several runnables with different priority levels. More generally, whenever task context-switches occur and data is shared between tasks, data consistency should be guaranteed.

For instance, variable x with an initial value 5 is shared by runnable A and B. A is going to calculate the expression x++ and send the result to runnable C. However, before A succeeds to write the result 6 to x, task-switching occurred, C added 2 to x and wrote 7 to x. Immediately, task switching occurrs again. After A retrieves the CPU, A wirtes the result 6 to x and sends its own result. Obviously, B will get 6, the update of runnable C will be lost.

To avoid data inconsistency, AUTOSAR suggests several mechanisms to protect “the critical data”, but in this paper, a new approach to avoid data inconsistency based on proper runnables mapping is proposed.

3.4. Time delay

While mapping multiple category 2 runnables with WaitPoint to the same task, different waitpoints may lead to communication delay. For example, if we put several runnables which have different DataReceive-Points and WaitPoints into the same task, even if a WaitPoint is resolved by an incoming RTEEvent, this task will still be waiting at a different WaitPoint. As a result, the communication will be delayed.

Taking into account these issues, mapping of runnables should be done carefully. We have to set up more rules to optimize the communication.

4. Mapping rules analysis

4.1. New rules to optimize intra-ECU communication

In order to optimize communication issues we have discussed above and get an efficient mapping of runnables, we extend the AUTOSAR basic rules by adding several new rules. However, it never means that we can break the AUTOSAR basic rules. Moreover, the newly added rules must not violate any AUTOSAR basic rules and they can only be used to map runnables in the same ECU. In other words, they work only for intra-ECU communication.

E.1) *If a runnable has only one DataSendPoint but no WaitPoint, and the corresponding receiver runnable has only one DataReceivePoint and at most one WaitPoint, then the sender and the receiver should be mapped to the same basic task.*

As can be seen from Figure 2, mapping the sender and receiver into two different tasks will cost one more task-switching than mapping them into the same one. In the former mapping case, it is necessary to guarantee data consistency of the global variable A, while in the latter case, we don't have data inconsistency problem thanks to the disappearance of task-switching.

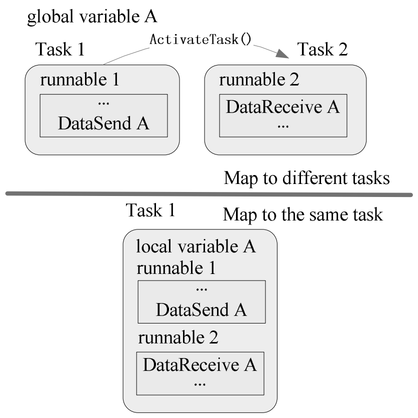

Figure 2. Map sender & receiver to the same task.

There is one more thing to note, that is, if runnable 1 has something complicated to do after it sends data A, runnable 2 will be delayed in the latter case. But we can ignore this issue in most cases in that runnables are generally small code fragments which cost a little time.

E.2) *If there are two or more runnables and each of them has only one DataReceivePoint which refers to the same data-element, these runnables should be mapped to the same task.*

Figure 3 illustrates the mapping rule E.2. Mapping N receivers to different tasks will cause at least N task-switching, however, it just cost one if they are put into the same task. Inevitably, data consistency must be guaranteed for the global variable A in the former case. In the latter case, a copy C should be created to ensure that the shared local variable A will not be modified by any runnables. Obviously, we could treat the runnables in task 2 as a runnable composition to further optimize the communication with rule E.1, provided that other conditions in the rule E.1 are satisfied.

E.3) *If two or more runnables implicitly use only DataReadAccess to receive the same data-element, then they should be mapped to the same basic task.*

In fact this rule is similar to E.2, but all runnables are in category 1, so it's better to map them to the same basic task.

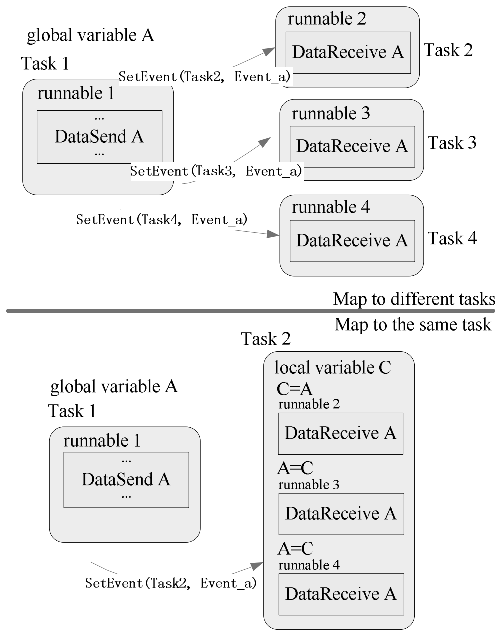

Figure 3. Map multiple receivers to the same task.

E.4) *If two or more runnables, which are activated by cyclic TimingEvents, each have only one DataSendPoint but no WaitPoint or implicitly use DataWriteAccess, try to send data to the same receiver, then these senders should be mapped into the same basic task.*

In this multiple senders' case, we map the senders, which run circularly, into the same basic task in order to reduce the task-switching time. The "isQueued" attribute of communication in this case should be set to true, which indicate that the received 'events' for the receiver have to be buffered in a queue [4].

Figure 4 shows the idea of rule E.4.

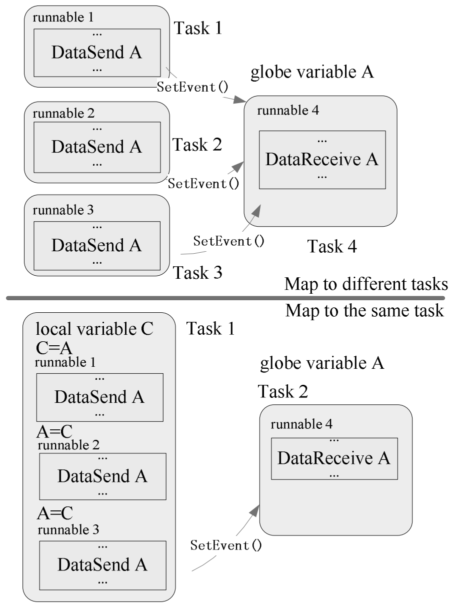

Figure 4. Map multiple senders to the same task.

E.5) *If the canBeinvokedConcurrently attribute of the server runnable is set and the server runnable is of category 1, then both the client and server runnable should be mapped to the same task*.

The RTE supports concurrent activation of the same instance of a runnable entity if canBeInvoked-Concurrently is set, which refer to reentrancy of the runnable. The server and the client can share the same stack without any problem. It's important to make sure that the client runnable should be run before the server is activated.

E.6) *If two or more runnables have different WaitPoints, then do not map them to the same task.*

Generally, in order to avoid time delay, it's a good practice to map two category 2 runnables with different WaitPoints in their own tasks.

In conclusion, it's actually a combinatorial optimization problem to assemble all the runnables in the ECU to OS tasks. These six rules help to reduce the task-switching, and avoid data inconsistency and time delay in intra-EUC communication.

4.2. Evaluate the mapping

Mapping runnables to tasks is an iterative process of computing, where we can not achieve the most efficient mapping just by performing the algorithm once. Essentially, it is an optimization problem of permutation and combination. In the case of 20 runnables, we usually have hundreds of mapping combination schemes to choose from. In order to choose the most efficient combination, the mapping performance can be evaluated by the following equation:

$$f_v = [0.05\frac{R}{2}F(E.6) + 0.01\sum_{n=1}^{5} C(E.n)]\prod_{i=1}^{3} T(B.i)$$

The value of f_v represents the evaluation result of the mapping. The greater f_v value is obtained, the better performance is achieved.

T(B.i) checks in every task whether AUTOSAR basic rule *B.i* has been violated, if none then return the result 1, otherwise 0. This means that none of the mappings can violate the AUTOSAR basic rules, otherwise the evaluation f_v will get its lowest value 0.

C(E.i) checks how many runnables satisfy the extended rule *E.i* in every task. For instance, if 7 runnables are mapped to the same task following the *E.1*, then *C(E.1)*=7. If there is only one receiver in the task then *C(E.1)*=0. We give rule *E.1* to *E.5* 0.01 weight.

Specially for rule *E.6*, *F(E.6)* check whether there are two or more different WaitPoints in the same task. If there is at most one then it returns 1, otherwise it returns 0. *R* represents the number of the runnables in the ECU. The coefficient *R/2* before *F(E.6)* means every two runnables that accord with *F(E.6)* will get a premium. Given that time delay will seriously affect the communication, we give rule *E.6* 0.05 weight.

When getting some mapping combinations, we should pay more attention to those combinations which get greater value of the evaluation equation. Because usually, some operations e.g. recombination can be performed on them to get better generations.

5. A mapping case for vehicle wheels control application

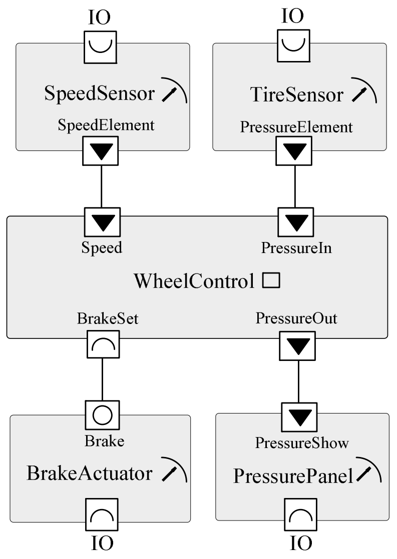

Figure 5. A vehicle wheel control application.

Figure 5 shows a vehicle wheel control application based on SmartOSEK OS. The speed sensor gets data element of speed from IO hardware abstraction layer and then provides the data through the SpeedElement port to Speed port. The WheelControl component gets the data and invokes the Brake operation via BrakeSet-Brake client-server communication. The BrakeActuator invokes IO hardware layer service to brake the car. TireSensor and PressurePanel act similarly as this Sensor-Control-Actuator model to show the pressure of the tire on the panel. Table 1，Table 2，Table 3，Table 4 and Table 5 show the runnables and their behaviors in the components.

Table 1. Runnables of SpeedSensor.

Runnable	SpeedGet	SpeedDenoising
Point	ServerCallPoint DataSendPoint	DataReceivePoint DataSendPoint
RTEEvent	TimingEvent	DataReceivedEvent

Table 2. Runnables of WheelControl.

Runnable	SpeedControl	PressureControl
Point	DataReceivePoint& WaitPoint serverCallPoint	DataReceivePoint& WaitPoint DataSendPoint
RTEEvent	DataReceivedEvent	DataReceivedEvent

Table 3. Runnables of BrakeActuator.

Runnable	BrakeSet
Point	serverCallPoint
RTEEvent	OperationInvokedEvent

Table 4. Runnables of TireSensor.

Runnable	PressureGet
Point	serverCallPoint DataSendPoint
RTEEvent	TimingEvent

Table 5. Runnable of PressurePanel.

component	PressurePanel
Runnable	PressureShow
Point	ServerCallPoint
RTEEvent	N/A

Three are plenty of mapping combinations of these runnables, we discuss three of them in Table 6(m.n means the corresponding runnable maps to task m and specifies its sequence to n if necessary).

Table 6. Three mappings.

Runnable name	1st	2nd	3rd
SpeedGet	1.1	1.1	1
SpeedDenoising	1.2	1.2	2
SpeedControl	1.3	2.1	3
BrakeSet	1.4	3	4
PressureGet	2.1	4	5
PressureControl	2.2	2.2	6
PressureShow	2.3	5	7

Table 7 below is the comparison of the mapping performance of those three mapping combinations. As can be seen from this table, the best choice is the first case. We count the task switching at the occurrence of the cyclic TimingEvent of SpeedGet on SmartOSEK OS. The first mapping just takes one task switching in every cycle of TimingEvent. And the communication delay time has been avoided. It's more efficient than other choices. Unfortunately, in the second mapping, two WaitPoints in task 2 lead to time delay.

Table 7. Comparison of the mappings.

Mapping No.	f_v	Task-switching	Delay time
1	0.25	1	no
2	0.06	6	yes
3	0.19	7	no

6. Conclusion and future work

In this paper, six extended rules and an evaluation equation to optimize the intra-ECU communication based on AUTOSAR have been proposed. In addition, we have given a study case to apply these rules and got an optimized result of mapping on SmartOSEK OS, which have a lowest task-switching and no delay time.

At last, there are several opportunities for future optimization on this issue:

- set up more efficient rules and more general evaluation equation;
- use formal language *Z* to express the mapping rules, test and verify the rules;
- improve the mapping program to get better mapping using genetic algorithm or other combinatorial optimization problems solution.

Acknowledgements

This research was supported by 863 National High Technology Program under Grant No. 2007AA01Z129 and 2007AA010304 and Key Program of Science and Technology Department of Zhejiang Province under Grant No. 2008C01036-3.

References

[1] AUTOSAR Partnership. Technical Overview V2.2.1 R3.0 Rev 0001. *http://www.autosar.org/*. February 2008.

[2] M. Zhao, Z. Wu, G Yang and L.Wang. SmartOSEK: A dependable platform for vehicle electronics. *The First International Conference on Embedded Software and System*, Vol.3605:437-444, December 2004.

[3] AUTOSAR Partnership. Specification of the Virtual Functional Bus V1.0.1 R3.0 Rev 0001. *http://www.autosar.org/*. February 2008.

[4] AUTOSAR Partnership. Specification of RTE V2.0.1 R3.0 Rev 0001. *http://www.autosar.org/*. February 2008.

[5] R. Racu，A. Hamann. Automotive Software Integration. *Design Automation Conference*, p545-550, June 2007.

[6] AUTOSAR Partnership. Specification of Operating System V3.0.1 R3.0 Rev 0001. *http://www.autosar.org/*.February 2008

[7] OSEK. OSEK/VDX Operating System Specification 2.2.3. *http://www.osek-vdx.org/*.

[8] K. Liu. Run-Time Environment Middleware for Vehicle Eletronics –SmartRTE. Master degree thesis, May 2008

[9] Z. Wu, H. Li, et al. An Improved Method of Task Context Switching in OSEK Operating System. *20th International Conference on Advanced Information Networking and Applications*, Vol.1: 217–222, April 2006.

[10] D. Schreiner. A Component Model for the AUTOSAR Virtual Function Bus. *International Computer Software and Applications Conference*, Vol2, p635-641, 2007.

[11] D. Kum, P. G. Min, et al. AUTOSAR Migration from Existing Automotive Software. *International Conference on Control, Automation and Systems*, p558-562, 2008

[12] H. Heinecke, W. Damm, et al. Software components for reliable automotive systems. *Design, Automation and Test in Europe*, p549-554, 2008

[13] H. Harald. Automotive system design-Challenges and potential. *Design, Automation and Test in Europe*, p656-657, 2005