ARTES

A network for Real-Time research and graduate Education in Sweden

Real-Time Graduate Student Conference 2005

Mälardalen University

UPPSALA UNIVERSITET

Swedish Foundation for Strategic Research

Preface

This volume contains the papers presented at the 5th ARTES Graduate Student Conference, held at Mälardalen University, from February 22-23, 2005.

As of today, 100 graduate students are active the ARTES network by registering as Real-Time Graduate Students. They thereby get access to the benefits provided by ARTES. This includes participation at the ARTES Summer School and ARTES funded PhD courses, mobility support provided by ARTES to new PhD students and public relation activities by ARTES industry ambassador to bring the research results to the public and industry. The present 100 real-time graduate students as well as the about 60 ARTES Real-Time doctors clearly indicate that a substantial amount of real-time research is conducted in Sweden. Due to ARTES and other efforts, it is fair to say that Sweden is one of the world-leaders in real-time systems research. We have an exiting meeting ahead of us.

The main idea with the ARTES Graduate Student Conference is to provide a forum for technical presentations and discussions among the Swedish graduate students active in the real-time area. For newly recruited graduate students it will provide an opportunity to experience "a real conference situation" (maybe) for the first time. For everyone, the conference will be an excellent opportunity to, in a relatively short time, get an overview of the current state of the national research. This year we have included a visit to ABB Robotics in co-operation with SAVE-IT. ARTES programme director from the start until 2004-12-31 will present the research at Mälardalen Real-Time Center and as a bonus the conference has been planned to end in conjunction with the start of the guest lectures by Wolfgang Weck and Mikael Åkerholm in the ARTES course Advanced Component Based Software Engineering.

The papers included in this volume indicate width and quality of Swedish Real-Time research. We are certain that the conference will be an event with intense technical and other discussion.

Enjoy it!

Paul Pettersson and Roland Grönroos
for the ARTES Programme
http://www.artes.uu.se/

PhD Student Kick-Off 2005
in Västerås
February 22-23

Schedule
Tuesday February 22

- 11.30 Lunch at Rosenhill, thereafter departure to ABB
- 13.00 ABB Robotics in collaboration with SAVE-IT
- 16.00 Session 1 in room R1-142. chair: Roland
  Paul Pettersson, A few words about ARTES
  Johan Erikson, The Parallel PLEX Project
  Jianlin Shi, Model Based Development and Competence Integration within Mechatronics
- 17.00 Break
- Session 2 in room R1-142. chair:
  Erik Kuiper, Robust Real-time Communication in Dynamic Wireless Ad Hoc Networks
  Viacheslav Izosimov, Design Optimization of Time- and Cost-Constrained Fault-Tolerant Distributed Embedded Systems
  Christer Gerdtman, Alternative input devices
- 19.30 Dinner at STRIKE, Torggatan 1

Wednesday February 23
Room: R1-142

- 9.00 **Hans Hansson et. al.** MRTC activities
- 10.00 Break with coffe
- 10.15 Session 3 chair:
  Najeem Lawal, Global Block RAM Allocation and Accesses for FPGA Implementation of Real-Time Video Processing Systems
  Niklas Lepistö, FPGA based Surveillance and Control Computer for Customer Specific Applications
  Fredrik Törner, Design of Electrical Architectures for Safety Cases
- 11.00 Session 4 chair:
  Johan Andersson, to be announced
  Pavel Krcal, REMODEL in Times
  John Håkansson, UML SPT in Times
- 12.00-13.00 Lunch
- 13.15 - 17.00 Possibility to attend the guest lectures in

the ADVANCED COMPONENT-BASED SOFTWARE ENGINEERING course.

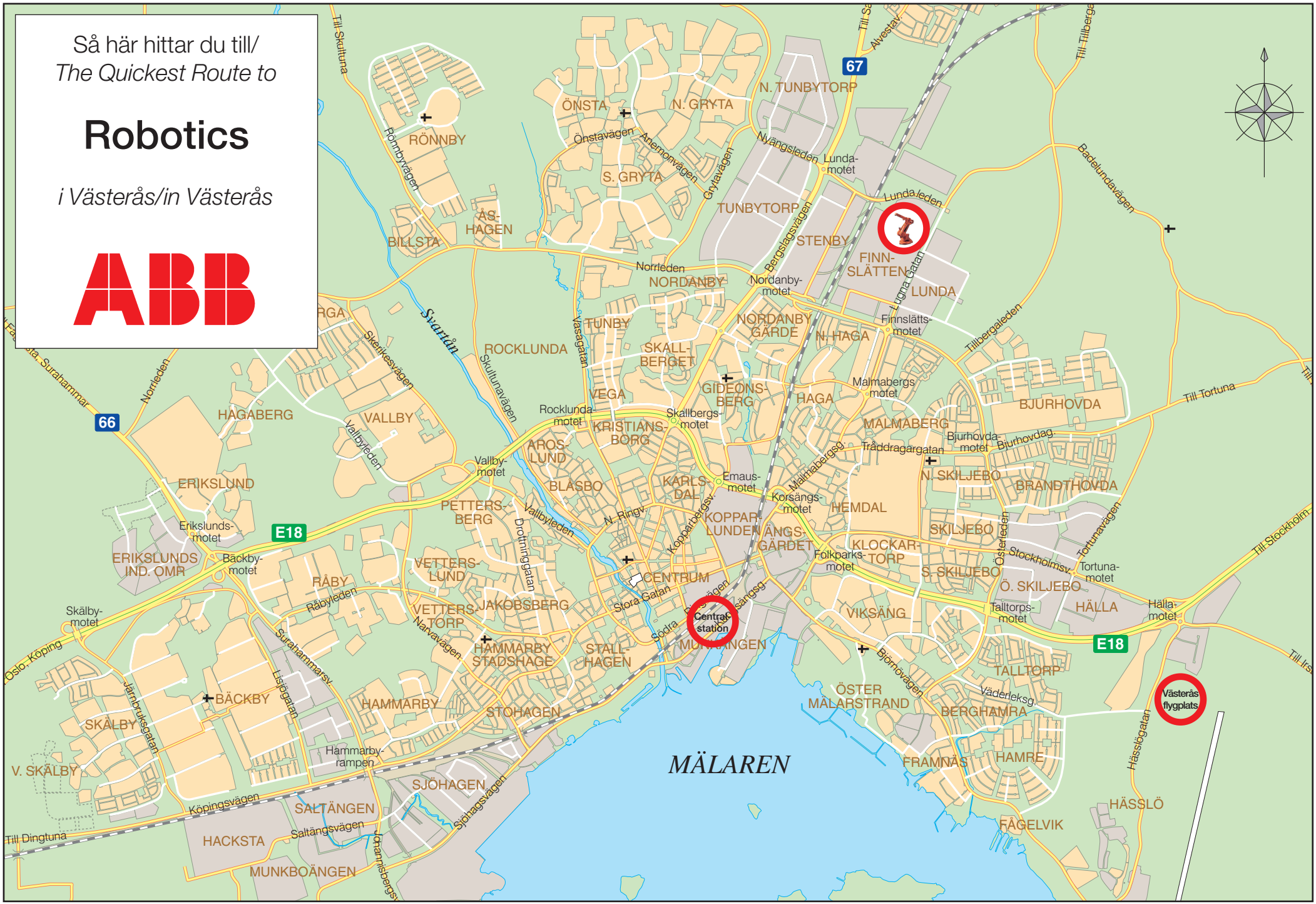
Wolfgang Weck - Eclipse Integration Framework
Mikael Åkerholm - SaveCCM Component model

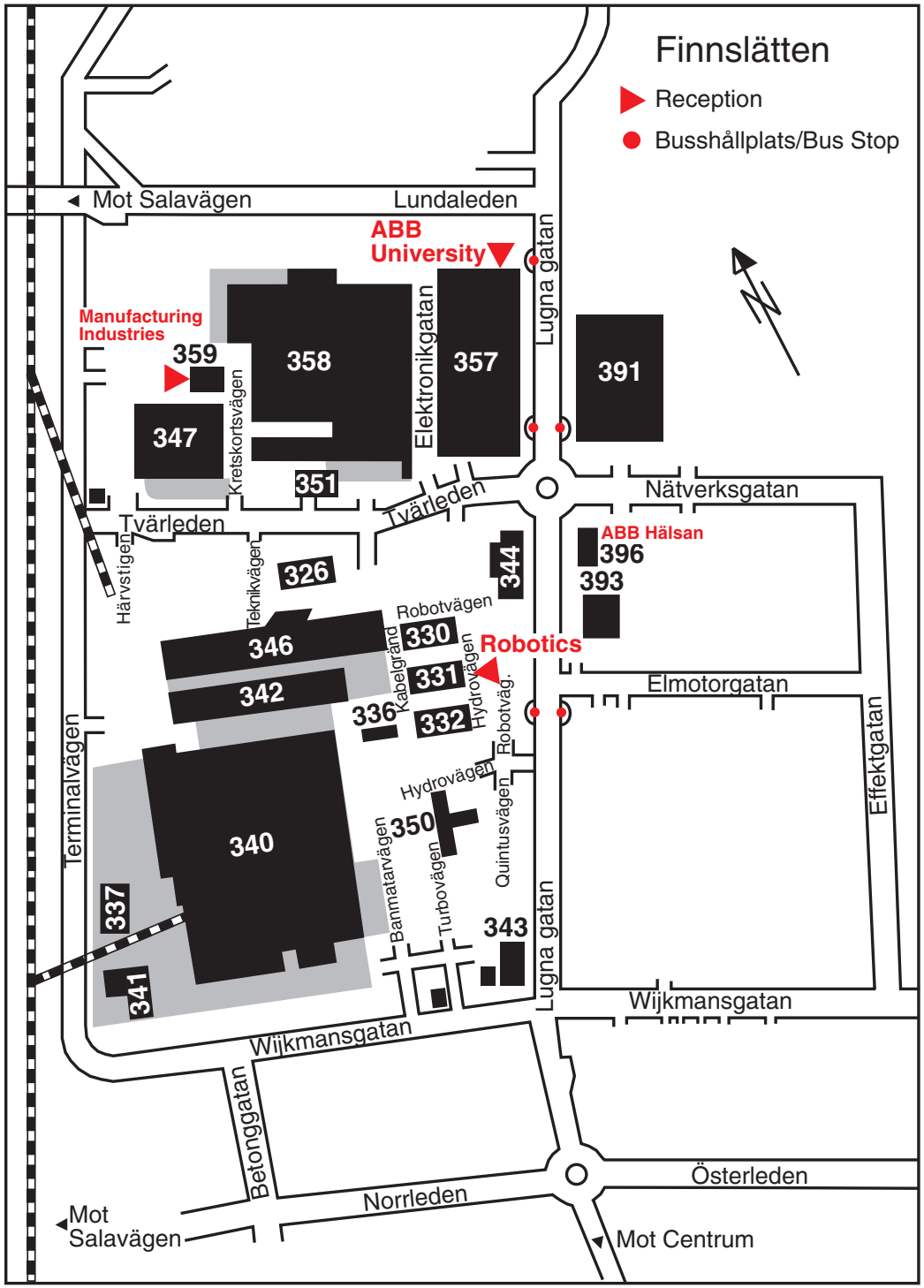
Updated: 21-Feb-2005 11:15
E-mail: artes@docs.uu.se Web: www.artes.uu.se
Location: http://www.artes.uu.se/events/gsconf05/schedule.shtml

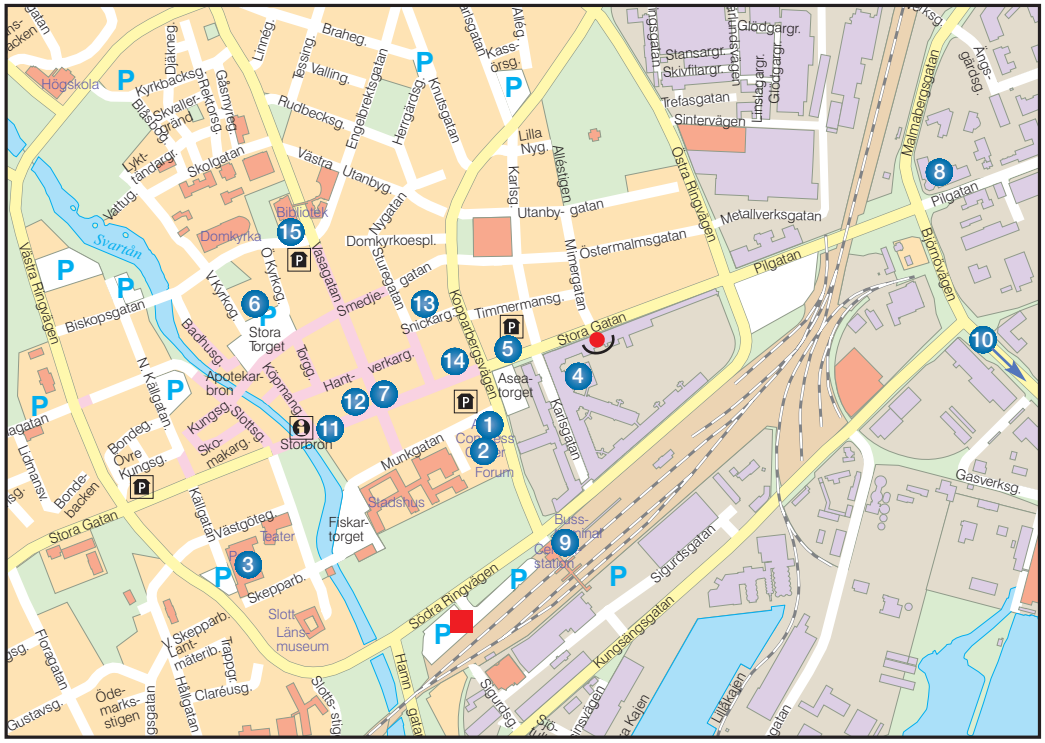
Så här hittar du till/
The Quickest Route to

Robotics

i Västerås/in Västerås







Finnslätten

ABB Automation Technology Products AB
Robotics har kontor /verkstad i byggnaderna
331, 330, 332, 346, 326, 342
Robotics has offices and workshops in buildings
331, 330, 332, 346, 326, 342

Robotics University building 357

ABB Automation Technology Products AB
Robotics
Hydrovägen 10
721 68 Västerås
tel 021-34 40 00

- Reception
  Reception
- Busshållplats
  Bus stop
- Buss till och från Arlanda: Bussterminalen vid järnvägsstation
  Arlanda - Västerås bus route: Bus terminal at the railway station and Scandic Hotel
- Buss från centrum till Finnslätten:
  Linje 11 från Stora Gatan
  City centre - Finnslätten
  Bus route: Use bus no. 11 from Stora Gatan

1. Aros Congress Center (ACC)
2. Konserhuset/*Concert hall*
3. Polisstation/*Police station*
4. ABB's huvudkontor/*ABB head office*
5. Radisson SAS Hotel
6. Stadshotellet
7. Comfort Hotel Etage
8. Scandic Hotel
9. Centralstation/*Railway station*
10. Västerås Flygplats/*Västerås Airport*
11. Turistbyrån/*Tourist information*
12. Apoteket/*Pharmacy*
13. Postkontor/*Post office*
14. Forex
15. Stadsbiblioteket/*Library*

Position Statement - The Parallel PLEX Project

Johan Erikson
Department of Computer Science and Electronics
Mälardalen University
johan.erikson@mdh.se
http://www.idt.mdh.se/personal/jen02

1 Background

The *Parallel PLEX Project* is a co-operation between Ericsson AB and the Department of Computer Science and Electronics at Mälardalen University. The project is facing the following general situation:

A complex legacy software system with independent jobs/tasks that executes in a non-preemptive, priority-based fashion. Since parallel processing was not an issue at the time the system was designed, programmers have assumed sequential execution and exclusive access to data, both in the design phase and (possibly years) later when the implementation has been updated.

The problem arises when the current single-processor architecture is to be replaced by a multi-processor ditto - *How is the system to be parallelized?* By simply moving the software from the old architecture to the new, it is most likely that the programs will break since independent, and concurrently executing, jobs may access and update the same data.

2 The AXE System

Our ”instance” of the general problem described in Section 1 is the AXE telephone exchange system, Fig 1, in general, and the language PLEX, which is used to program the functionality in central parts of the AXE system, in particular. (PLEX is covered in Section 3.)

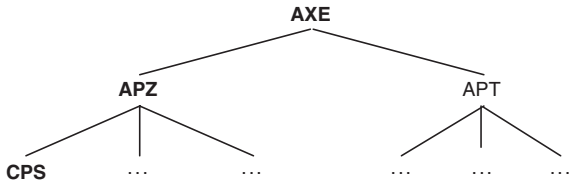


Figure 1: *Structure of the AXE telephone exchange system. PLEX is used to program the Central Processor Sub-system, CPS.*

The AXE system, developed in its earliest version in the beginning of the 1970s, is structured in a modular and hierarchical way. It consists of the two main parts:

APT: The telephony or switching part

APZ: The control part including central and regional processors

which both consist of hardware **and** software. The two main parts are divided into subsystems, where the part that is of interest for us is the *Central Processor Subsystem* (CPS), see Fig. 1.

In the current architecture, the control system (APZ) consists of a *Central Processor* (CP) (which in turn consists of a single *CPU* and additional software) and a number of *Regional Processors* (RP), see Fig. 2. Call requests are received by the RP's, inserted into job buffers, and processed by the CP. Due to the "pseudo-parallel" structure of PLEX (described in the following section), parallel processing seems like a natural choice to increase performance and through-put in the AXE system[1], but as will be discussed in Section 4, we are facing exactly the same problem with the softare as was described in Section 1.

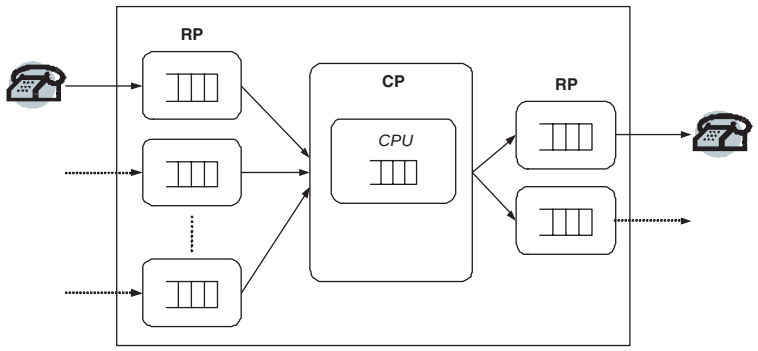


Figure 2: *Call requests received by Regional Processors (RP) and processed by the Central Processor (CP).*

3 PLEX, Programming Language for EXchanges

The language PLEX, is a pseudo-parallel and event-driven real-time language developed by *Ericsson*. The language has a signal paradigm as its top execution level, and it is event-driven in the sense that only events, encoded as signals, can trigger code execution. A typical event is an incoming *call request* (see Fig. 2).

A PLEX program file (called a *block*) is divided in several, independent *sub-programs*, see Fig. 4 (a), which can be executed in any order. One or several sub-programs constitutes a *Job*, which is a continuous sequence of statements executed in the processor. This means that the execution of a PLEX program consists of the execution of a number of independent and "parallel" jobs. However, the jobs are not executed truly in parallel: rather, when spawned, they are put in one of four queues, of different priority, and sequentially executed in a non-preemptive fashion, Fig 3, thus the term "pseudo-parallel". Jobs communicate and control other jobs by the usage of signals.

The entry points of the sub-programs are the only entry points to a block. Variables are declared in a *common data area*, and they have their scope inside the block they are declared in. It is not possible to access variables from another block, except through via one of the sub-programs (in that block).

[1] A thorough description of the AXE system, as well as of the PLEX language (which we only cover briefly in Section 3), is given in [EL02]

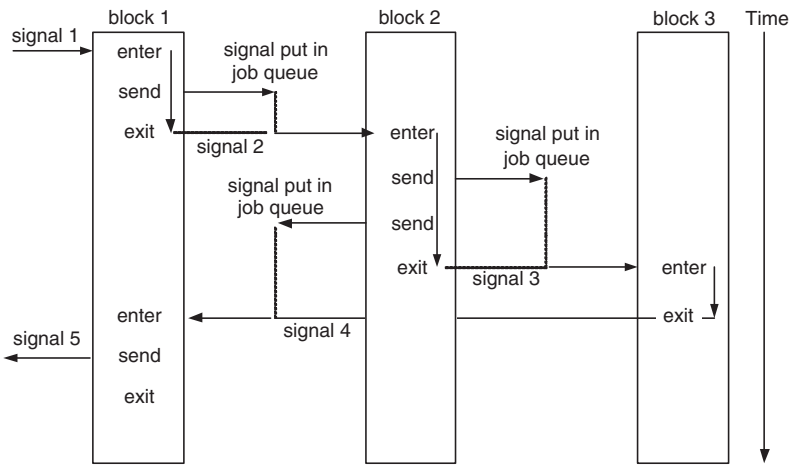


Figure 3: *The execution model of PLEX.*

Due to the information hiding and data encapsulation, PLEX blocks can be thought of as objects, and therefore PLEX may be seen as an early object-based language, see Fig. 4 (a).

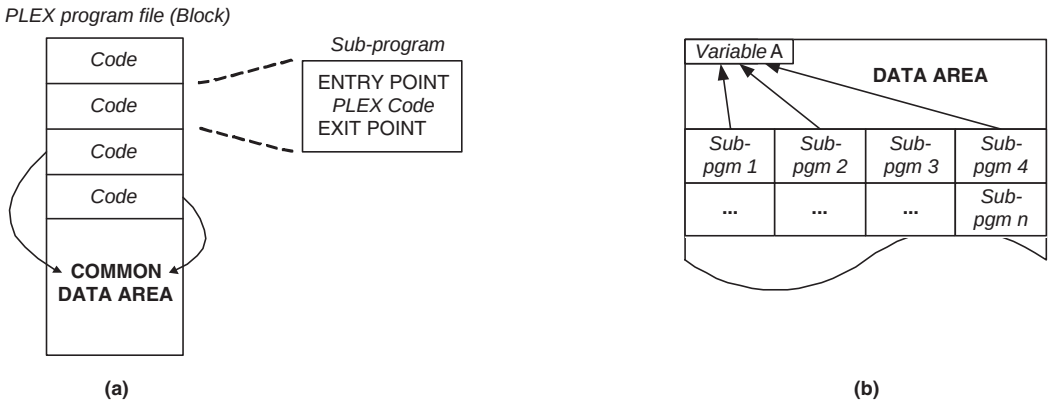


Figure 4: *PLEX blocks: could be seen as objects (a), but also with potential conflicts (b).*

4 The Problem

In the current single-processor architecture, there can be no conflicts between the independent sub-programs that access the same data area since they are executed in sequence. But, moving to a multi-processor environment, where new jobs would be forked off as parallel threads instead of being buffered, introduces the following problem: several, independent, jobs may execute code (sub-programs) from the same block, where the sub-programs access the same data, see Fig. 4 (b).

The problem could for instance be solved by (1) extending PLEX with mechanisms to protect shared variables (e.g., with a lock), rewrite and recompile the code; or (2) replace PLEX

with a language originally designed for parallel execution, and then rewrite the system.

However, these solutions have the following drawbacks:

- First of all - The AXE system consists of 20 Mlines of PLEX code! This fact makes any attempt to rewrite the entire system impossible due to time and money.
- It will probably be hard to find a language with a similar execution model as PLEX - which would be an absolute necessity if PLEX is to be replaced.

  There have been attempts to introduce other languages, like C++, and let the system execute PLEX code as well as C++ code. However, these attempts have failed, most likely due to the different execution models.
- Even if a suitable language (to replace PLEX with) would be found, one would have to decide on the following; (1) Either the language has to be backward compatible with PLEX to prevent rewriting the entire system; or (2) existing customers would have to re-invest in a new system since their existing equipment would not be compatible with the new.

5 Our Approach

We believe that the only practical solution is to find methods that decides when parallel execution of the current software is *safe*, i.e., when parallel execution of jobs yields the same behavior as sequential execution without resulting in data interference, Fig. 4 (b).

We don't think that the need to add synchronization primitives to **some** blocks (and then re-compile them) can be totally eliminated, but our goal is to substantially reduce the number of blocks that has to be modified in this way.

Our approach is to use program analysis to distinguish between PLEX blocks that can / can not execute in parallel, and verify the analysis with formal semantics.

- The program analysis phase will reveal how variables are read from / written to, and also if two jobs interfere in, or communicate through a variable. The first part (read/write) is already implemented in the PLEX compiler [AE00], whereas the second has been discussed by Lindell [Lin03].

  These steps will allow us to decide on whether two jobs can execute in parallel or not, which will be the basis for the run-time scheduling.
- The formal semantics, which is a conventional structural operational semantics in the style used in [NN92], will be developed in three steps.

  1. The first step is to define the semantics for sequential execution of PLEX in the current single-processor architecture, i.e., a specification/formalization of the behavior of PLEX in the system as it is today. This step has been presented as a Technical Report [Eri03], as well as at a workshop on *Applied Semantics* [EL04].
  2. The second step will specify how PLEX should behave in a chosen multi-processor paradigm **without** changes in the language, i.e., without synchronization primitives.
  3. The last step will specify the semantics for "Parallel-PLEX", i.e., PLEX with additional synchronization primitives.

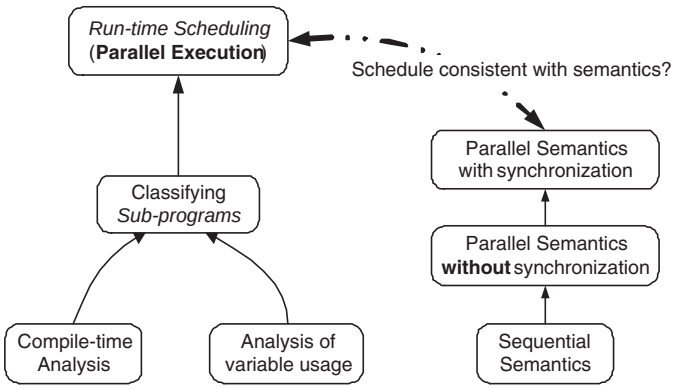


Figure 5: *The approach to solve the problem.*

The semantics will then be used to verify the correctness of the decisions taken in the program analysis phase. The approach, with the single sub-steps, are shown in Fig. 5.

Besides ensuring the correctness of parallel execution, we also aim at being able to state rules for how to (in a first step) adapt PLEX programs manually for parallel execution, and (in a second step) automate this with the program analysis.

6 A Formal Semantics for PLEX

If we look at Fig. 5, our work up til today has mainly been focused on developing the formal semantics for PLEX, why we end this report with a brief summary of the semantics.

As stated earlier (Section 5), we have chosen a conventional structural operational semantics to model the behavior of PLEX, where the execution of statements is modeled by state transitions of the form

$$\langle S,\ s\rangle\ \Rightarrow\ s'$$

i.e., the execution of the statement S from an initial state s results in a new state s'. This means that the key thing at this point is to define the *state of the system* which is to be modeled, and since PLEX allows different kinds of unstructured jumps, we first introduce a *virtual statement counter*, $\mathcal{VSC}$, which identifies the current statement to be executed. This counter is made explicit in the program state and is used in the following way:

$$\langle \texttt{GOTO}\ label,\ s\rangle\ \Rightarrow\ s[\mathcal{VSC} \mapsto \mathcal{ADR}[\![label]\!]]$$

i.e., the execution of the `GOTO` statement, from the initial state s, results in a new state where the next statement to be executed is the statement found at $label$.

Apart from this counter, the state is defined by the contents in the memory, including different data areas and job queues.

When we move to a multi-processor architecture (with k processors), the state transitions will have the form

$$\langle S,\ i,\ s\rangle\ \Rightarrow\ s'$$

where we now has made explicit **where** the statement is executed (on processor P_i).

At this stage, the processors are made explicit in the system state, and with | as our parallelizing operator, we use the notation $P_i|P_j$ to denote that processors P_i and P_j may execute in parallel.

The above leads to following, global, state transitions

$$\frac{\langle S,\ i,\ s\rangle\ \Rightarrow\ s'}{\langle \ldots \mid P_i \mid \ \ldots\ ,\ s_i\ \ldots,\ s_G\rangle \Rightarrow \langle \ldots \mid P_i' \mid \ \ldots\ ,\ s_i'\ \ldots,\ s_G\rangle}$$

to capture that there is a global change in the state

$$\langle \ldots \mid P_i \mid \ \ldots\ ,\ s_i\ \ldots,\ s_G\rangle \Rightarrow \langle \ldots \mid P_i' \mid \ \ldots\ ,\ s_i'\ \ldots,\ s_G\rangle$$

if there is a valid, local transition of the form $\langle S,\ i,\ s\rangle\ \Rightarrow\ s'$

References

[AE00] J. Axelsson and J. Erikson. SAPP, Theories and Tools for Execution Time Estimation for Soft Real Time (Communication) Systems. Master's thesis, Mälardalen University, 2000.

[EL02] J. Erikson and B. Lindell. The Execution Model of the APZ/PLEX - An Informal Description. Technical report, Mälardalen University, 2002.

[EL04] J. Erikson and B. Lisper. A formal semantics for PLEX. In *Proceedings of the 2nd APPSEM II Workshop, APPSEM'04*, Tallin, 14-16 April 2004.

[Eri03] J. Erikson. A Structural Operational Semantics for PLEX. MRTC Report, ISSN 1404-3041 ISRN MDH-MRTC-166/2004-1-SE, Mälardalen University, 2003.

[Lin03] B. Lindell. Analysis of reentrancy and problems of data interference in the parallel execution of a multi processor AXE-APZ system. Master's thesis, Mälardalen University, 2003.

[NN92] H. R. Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*. John Wiley & Sons, 1992.

Model Based Development and Competence Integration within Mechatronics - An introduction of research topic

Jianlin Shi

KTH, Machine Design

Background

In the modern industry where mechatronic systems are widely adopted more and more functionalities are highly dependent on software and electrical components apart from traditional pure mechanical components. This highly increased the complexity and requires a mature engineering method for modeling and analysing in an integrated way for the entire system and subsystems. The key problem includes both efficient technology integration and competence integration.

Purpose and aim

This project is aimed to improve management of the complexity within the development of mechatronic system products by developing knowledge, support methods and prototype tools. The status of problems is attacked from two supplemental approaches:

a. *How to manage technology integration within mechatronic development?*
b. *How to solve the competence integration over disciplines boundaries within mechatronic development?*

The project is cooperation between two groups in KTH, Mechatronic and Integrate Product Development. My responsibility mainly focus on part a. However, both parts are treated jointly in collaboration with a number of other researchers. Expected contributions include state of practice documentation and analysis, requirements specification, development of modelling framework and prototype tools.

Central hypothesis

Model-based development (MBD), here defined as "*systematic use of modelling throughout the life cycle to support product development and maintenance using adequate tools*", where models are compatible over domain boundaries and provides possibilities for integrated system analysis, is an essential approach for efficient development of mechatronic products. It is also the prerequisite for the necessary competence integration, and therefore makes better use of developments resources.

State of art and approaches:

Due to the integration of multiple domains, the integrations of tools and information become the two main topics. Current researches of model and tool integration are code generation, co-simulation and model import/export. The Object Management Group has promoted a new MBD approach "Model-Driven Architecture" (MDA), which adopts number of technologies such as UML, MOF, specific models, SysML and UML profiles to provide high portability, interoperability and reusability through architectural separation of concerns. Since the UML 2.0 improves it precision and expressiveness to support component-based development, internal and cross integration structural and behaviour as well as integration of action semantics, it becomes more practicable for industry combined with other modelling tools. On another way, aiming to represent products without loss of complexity and integrity, STEP (the Standard for the Exchange of Product Model Data) provides an ISO standard that describes how to represent and exchange product information during the life cycle.

The project is planed to be performed in five steps as described in following: (1) Case study of current situation at VCC, (2) Literature review and development of modelling framework which contains the most important subsystems and characteristics, (3) methodology development, (4) new work methods implementation together with (5) test, evaluate and analyse generated results and eventually draw conclusions.

Status: The project was started in late of 2004 and currently step 1 and step 2 are on going.

Erik Kuiper

Robust Real-time Communication in Dynamic Wireless Ad Hoc Networks

Who am I

- MSc in Industrial Engineering and Management
- Have been working at Saab for 4 years
  - 3.5 years of software development
  - 0.5 years of technology studies
- Live in Linköping

The System

- Independent mobile nodes
- Nodes can be added and removed from the system at any time
- Communicate over a wireless ad hoc network
- Real-time requirements on the communication
- Limited bandwidth
- Varying amount of bandwidth available
- Non-exclusive media usage

The Question

- How to allocate the limited and varying bandwidth in real-time
- How to communicate available network services
- How to handle that nodes are added to and removed from the network "stochastically"

Excluded Issues

- Security

Research Method

- Determine current state of the art
- Develop communication protocols
- Test communication protocols by simulation
- Iterate and improve

Design Optimization of Time- and Cost-Constrained Fault-Tolerant Distributed Embedded Systems

Viacheslav Izosimov, Paul Pop, Petru Eles, Zebo Peng
Computer and Information Science Dept., Linköping University, Sweden
{viaiz, paupo, petel, zebpe}@ida.liu.se

Abstract

In this paper we present an approach to the design optimization of fault-tolerant embedded systems for safety-critical applications. Processes are statically scheduled and communications are performed using the time-triggered protocol. We use process re-execution and replication for tolerating transient faults. Our design optimization approach decides the mapping of processes to processors and the assignment of fault-tolerant policies to processes such that transient faults are tolerated and the timing constraints of the application are satisfied. We present several heuristics which are able to find fault-tolerant implementations given a limited amount of resources. The developed algorithms are evaluated using extensive experiments, including a real-life example.

1. Introduction

An increasing number of embedded applications require high levels of dependability. For example, the automotive industry requires very low failure rates of 10^{-9} failures/hour [12]. In many application areas, including the automotive industry, application-specific fault tolerance methods are currently used, which rely on reasonableness checks based on application knowledge, interwined with the application code. On the other hand, systematic fault-tolerance techniques, are based on the replication of components and are transparent to the application. Due to the significant decrease in semiconductor costs, and the increase in complexity, systematic-fault tolerance approaches are more and more preferred [12, 16]. However, such systematic approaches increase the costs of an embedded system, which makes them, for the moment, inapplicable to a large range of cost-sensitive application areas. Cost-effective systematic fault-tolerance is therefore needed for increasing the dependability levels of safety-critical applications implemented on cost-constrained embedded systems.

Safety-critical applications have to function correctly and meet their timing constraints even in the presence of faults. Such faults can be permanent (i.e., damaged micro-controllers or communication links), transient (e.g., caused by electromagnetic interference), or intermittent (appear and disappear repeatedly). The transient faults are the most common, and their number is continuously increasing due to the continuously raising level of integration in semiconductors.

Researchers have proposed several hardware architecture solutions, such as MARS [13], TTA [14] and XBW [4], that rely on hardware replication to tolerate a single permanent fault in any of the components of a fault-tolerant unit. Such approaches, can be used for tolerating transient faults as well, but they incur very large hardware cost. An alternative to such purely hardware-based solutions are approaches such as re-execution, replication, checkpointing.

Pre-emptive on-line scheduling environments are flexible enough to handle such fault-tolerance policies. Several researchers have shown how the schedulability of an application can be guaranteed at the same time with appropriate levels of fault-tolerance [1, 2, 8, 17]. However, such approaches lack the predictability required in many safety-critical applications, where static off-line scheduling is the only option for ensuring both the predictability of worst-case behavior, and high resource utilization [12].

The disadvantage of static scheduling approaches, however, is their lack of flexibility, which makes it difficult to integrate tolerance towards unpredictable fault occurrences. Thus, researchers have proposed approaches for integrating fault-tolerance into the framework of static scheduling. A simple heuristic for combining together several static schedules in order to mask fault-patterns through replication is proposed in [5], without considering the timing constraints of the application. This approach is used as the basis for cost and fault-tolerance trade-offs within the Metropolis environment [15]. Graph transformations are used in [3] in order to introduce replication mechanisms into an application. Such a graph transformation approach, however, does not work for re-execution, which has to be considered during the construction of the static schedules.

Fohler [6] proposes a method for joint handling of aperiodic and periodic processes by inserting slack for aperiodic processes in the static schedule, such that the timing constraints of the periodic processes are guaranteed. In [7] he equates the aperiodic processes with fault-tolerance techniques that have to be invoked on-line in the schedule table slack to handle faults. Overheads due to several fault-tolerance techniques, including replication, re-execution and recovery blocks, are evaluated.

When re-execution is used in a distributed system, Kandasamy [10] proposes a list-scheduling technique for building a static schedule that can mask the occurrence of faults, thus making the re-execution transparent. Slacks are inserted into the schedule in order to allow the re-execution of processes in case of faults. The faulty process is re-executed, and the processor switches to a contingency schedule that delays the processes on the corresponding processor, making use of the slack introduced. The authors propose an algorithm for reducing the necessary slack for re-execution. This algorithm has later been applied to the fault-tolerant transmission of messages on a time-division multiple-access bus (TDMA) [11].

Applying such fault-tolerance techniques introduces overheads in the schedule and thus can lead to unschedulable systems. Very few researchers [10, 15] consider the optimization of implementations to reduce the overheads due to fault-tolerance and, even if optimization is considered, it is very limited and does not include the concurrent usage of

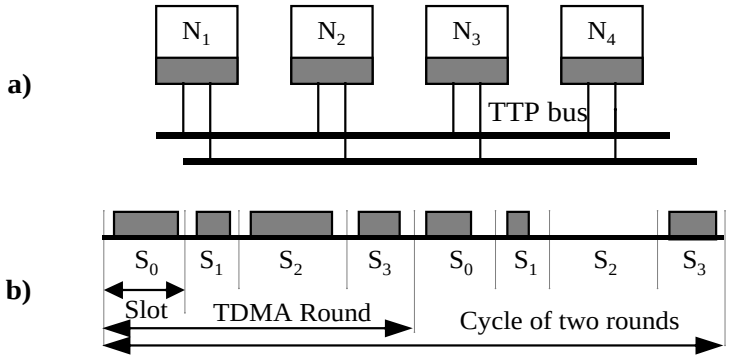


Figure 1. A System Architecture Example

several fault-tolerance techniques. Moreover, the application of fault-tolerance techniques is considered in isolation, and thus is not reflected at all levels of the design process, including mapping, scheduling and bus access optimization. In addition, the communication aspects are not considered or very much simplified.

In this paper, we consider hard real-time safety-critical applications mapped on distributed embedded systems. Both the processes and the messages are scheduled using static cyclic scheduling. The communication is performed using a communication environment based on the time-triggered protocol [13]. We consider two distinct fault-tolerance techniques: re-execution of processes, which provides time-redundancy, and active replication, which provides space-redundancy. We show how re-execution and active replication can be combined in an optimized implementation that leads to a schedulable fault-tolerant application without increasing the amount of employed resources. We propose several optimization algorithms for the mapping of processes to processors and the assignment of fault-tolerance techniques to processes such that the application is schedulable and no additional hardware resources are necessary.

The next two sections present the system architecture and the application model, respectively. Section 4 introduces the design optimization problems tackled, and Section 5 proposes a tabu-search based algorithm for solving these problems. The evaluation of the proposed approaches, including a real-life example consisting of a cruise controller are presented in Section 6. The last section presents our conclusions.

2. System Architecture

2.1 Hardware Architecture and Fault Model

We consider architectures composed of a set N nodes which share a broadcast communication channel. Every *node* $N_i \in N$ consists, among others, of a communication controller and a CPU. Figure 1a depicts an architecture consisting of four nodes.

The communication controllers implement the protocol services and run independently of the node's CPU. We consider the time-triggered protocol (TTP) [13] as the communication infrastructure for a distributed real-time system. However, the research presented is also valid for any other TDMA-based bus protocol that schedules the messages statically based on a schedule table like, for example, the SAFEbus [9] protocol used in the avionics industry.

The TTP has a replicated bus that integrates all the services necessary for fault-tolerant real-time systems. According to the TTP, each node N_i can transmit only during a predetermined time interval, the so called TDMA slot S_i, see Figure 1b. In such a slot, a node can send several messages packed in a frame. A sequence of slots corresponding to all the nodes in the TTC is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table for the TTP controller which has to know when to send/receive a frame to/from the communication channel.

In this paper we are interested in fault-tolerance techniques for tolerating transient faults, which are the most common faults in today's embedded systems. We have generalized the fault-model from [10] that assumes that one single transient fault may occur on any of the nodes in the system during the application execution. In our model, we consider that at most k transient faults[1] may occur anywhere in the system during one operation cycle of the application. Thus, not only several transient faults may occur simultaneously on several processors, but also several faults may occur on the same processor. We consider that the transient faults can have a worst-case duration of μ, from the moment the fault is detected until the system is back to its normal operation, and that a fault is confined to a single process and does not affect other processes.

2.2 Software Architecture and Fault-Tolerance Techniques

We have designed a software architecture which runs on the CPU in each node, and which has a real-time kernel as its main component. The processes are activated based on the local schedule tables, and messages are transmitted according to the MEDL. For more details about the software architecture and the message passing mechanism the reader is referred to [20].

1. The number of faults k can be larger than the number of processors in the system.

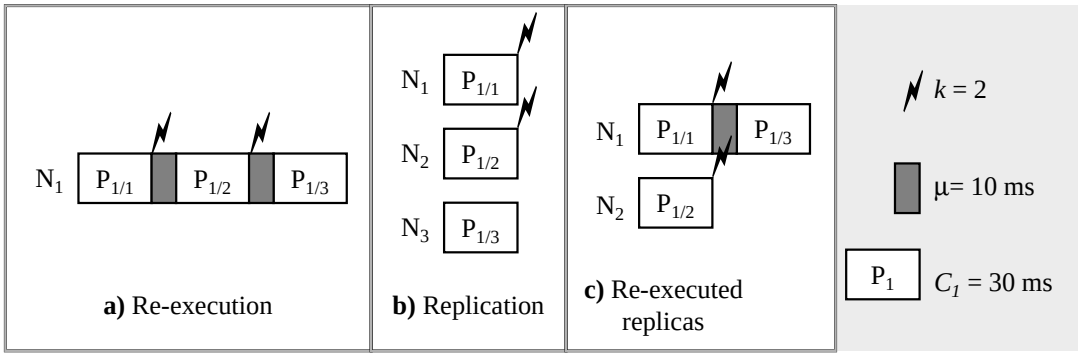


Figure 2. Worst-Case Fault Scenario and Fault-Tolerant Techniques

The error detection and fault-tolerance mechanisms are part of the software architecture. We assume a combination of hardware-based (e.g., watchdogs, signature checking) and software-based error detection methods, systematically applicable without any knowledge of the application (i.e., no reasonableness and range checks) [4]. We also assume that all faults can be found using such detection methods, i.e., no byzantine faults which need voting on the output of replicas for detection. The software architecture, including the real-time kernel, error detection and fault-tolerance mechanisms are themselves fault-tolerant.

We use two mechanisms for tolerating faults: re-execution and active replication. Let us consider the example in Figure 2, where we have process P_1 and a fault-scenario consisting of $k = 2$ faults with a duration $\mu = 10$ ms that can happen during one cycle of operation. In Figure 2a we have the worst-case fault scenario for re-execution, when the first fault happens at the end of the process P_1's execution. The fault is detected and, after an interval μ, P_1 can be re-executed. Its second execution is labeled with $P_{1/2}$, which, in the worst-case could also experience a fault at the end. Finally, the third re-execution of P_1, namely $P_{1/3}$, will execute without error. In the case of active replication, depicted in Figure 2b, each replica is executed on a different processor. Three replicas are needed to tolerate the two possible faults and, in the worst-case scenario depicted in Figure 2b, only the execution of $P_{1/3}$ is successful. In addition, we consider a third case, presented in Figure 2c, which combines re-execution and replication for tolerating faults in a process. In this case, for tolerating the two faults we use two replicas and one re-execution: the process $P_{1/1}$, which has $P_{1/2}$ as a replica, is re-executed.

With active replication, the input has to be distributed to all the replicas. Since we do not consider the type of faults that need replica agreement, our execution model assumes that the descendants of replicas can start as soon as they have received the first valid message from a replica. Replica determinism is achieved as a by-product of the underlying TTP architecture [16].

3. Application Model

We model an application A as a set of directed, acyclic, polar graphs $G(V, E) \in A$. Each node $P_i \in V$ represents one process. An edge $e_{ij} \in E$ from P_i to P_j indicates that the output of P_i is the input of P_j. A process can be activated after all its inputs[1] have arrived and it issues its outputs when it terminates. The communication time between processes mapped on the same processor is considered to be part of the process worst-case execution time and is not modeled explicitly. Communication between processes mapped to different processors is performed by message passing over the bus. Such message passing is modeled as a communication process inserted on the arc connecting the sender and the receiver process.

The combination of fault-tolerance policies to be applied to each process is given by two functions. F_R: $V \rightarrow V_R$ determines which processes are replicated. When active replication is used for a process P_i, we introduce several replicas into the process graph G, and connect them to the predecessors and successors of P_i. The second function F_X: $V \cup V_R \rightarrow V_X$ applies re-execution to the processes in the application, including to the replicas in V_R, if necessary, see Figure 2c. Let us denote the tuple $<F_R, F_X>$ with F.

The mapping of a process graph G is given by a function M: $V \cup V_R \rightarrow N$, where N is the set of nodes in the architecture. For a process $P_i \in V \cup V_R$, $M(P_i)$ is the node to which P_i is assigned for execution. Each process P_i can potentially be mapped on several nodes. Let $N_{P_i} \subseteq N$ be the set of nodes to which P_i can potentially be mapped. We consider that for each $N_k \in N_{P_i}$, we know the worst-case execution time $C_{P_i}^{N_k}$ of process P_i, when executed on N_k. We also consider that the size of the messages is given.

All processes and messages belonging to a process graph G_i have the same period $T_i = T_{G_i}$ which is the period of the process graph. A deadline $D_{G_i} \leq T_{G_i}$ is imposed on each process graph G_i. In addition, processes can have associated individual release times and deadlines. If communicating processes are of different periods, they are combined into a hyper-graph capturing all process activations for the hyper-period (LCM of all periods).

4. Design Optimization Problems

In this paper, by policy assignment we denote the decision whether a certain process should be re-executed or replicated. Mapping a process means placing it on a particular node in the architecture.

1. As already noted, the first valid message from the replicas is considered as the input.

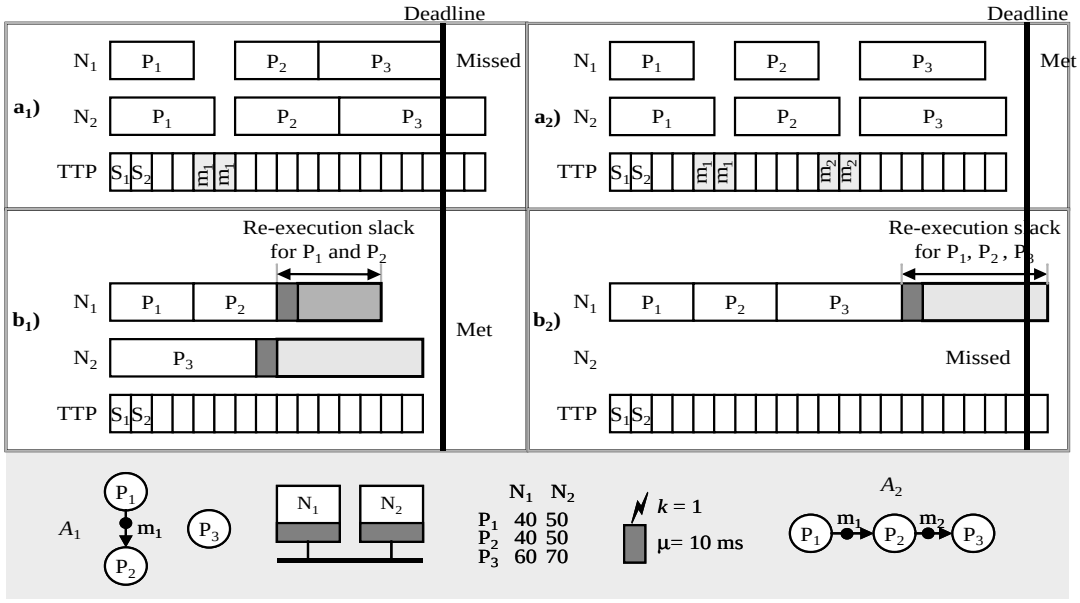


Figure 3. Comparison of Replication and Re-Execution

There could be cases where the policy assignment decision is taken based on the experience and preferences of the designer, considering aspects like the functionality implemented by the process, the required level of reliability, hardness of the constraints, legacy constraints, etc. We denote with P_R the subset of processes which the designer has assigned replication, while P_X contains processes which are to be re-executed.

Most processes, however, do not exhibit certain particular features or requirements which obviously lead to re-execution or replication. Let P be the set of processes in the application A. The subset $P^+ = P \setminus (P_X \cup P_R)$ of processes could use any of the two techniques for tolerating faults. Decisions concerning the policy assignment to this set of processes can lead to various trade-offs concerning, for example, the schedulability properties of the system, the amount of communication exchanged, the size of the schedule tables, etc.

For part of the processes in the application, the designer might have already decided their mapping. For example, certain processes, due to constraints like having to be close to sensors/actuators, have to be physically located in a particular hardware unit. They represent the set P_M of already mapped processes. Consequently, we denote with $P^* = P \setminus P_M$ the processes for which the mapping has not yet been decided.

Our problem formulation is as follows:

- As an input we have an application A given as a set of process graphs (Section 3) and a system consisting of a set of nodes N connected using the TTP.
- The fault model is given by the parameters k and μ, which denote the total number of transient faults that can appear in the system during one cycle of execution and their duration, respectively.
- As introduced previously, P_X and P_R are the sets of processes for which the fault-tolerance policy has already been decided. Also, P_M denotes the set of already mapped processes.

We are interested to find a system configuration ψ such that the k transient faults are tolerated and the imposed deadlines are guaranteed to be satisfied, within the constraints of the given architecture N.

Determining a system configuration $\psi = \langle F, M, S \rangle$ means:

1. finding a combination of fault-tolerance policies F for each processes in $P^+ = P \setminus (P_R \cup P_X)$;
2. deciding on a mapping M for each processes in $P^* = P \setminus P_M$;
3. deriving the set S of schedule tables on each processor and the MEDL for the TTP.

4.1 Fault-Tolerance Policy Assignment

Let us illustrate some of the issues related to policy assignment. In the example presented in Figure 3 we have the application A_1 with three processes, P_1 to P_3, and an architecture with two nodes, N_1 and N_2. The worst-case execution times on each node are given in a table to the right of the architecture. Note that N_1 is faster than N_2. The fault model assumes a single fault, thus $k = 1$, with a duration μ = 10 ms. The application A_1 has a deadline of 140 ms depicted with a thick vertical line. We have to decide which fault-tolerance technique to use. In Figure 3 we depict the schedules[1] for each node, and for the TTP bus. Node N_1 is allowed to transmit in slot S_1, while node N_2 can use slot S_2. A TDMA round is formed of slot S_1 followed by slot S_2, each of 10 ms length. Comparing the schedules in Figure $3a_1$ and $3b_1$, we can observe that using (a_1) active replication the deadline is missed. In order to guarantee that time constraints are satisfied in the presence of faults, re-execution slacks have to finish before the deadline. However, using (b_1) re-execution we are able to meet the deadline. However, if we consider application A_2 with process P_3

1. The schedules depicted are optimal.

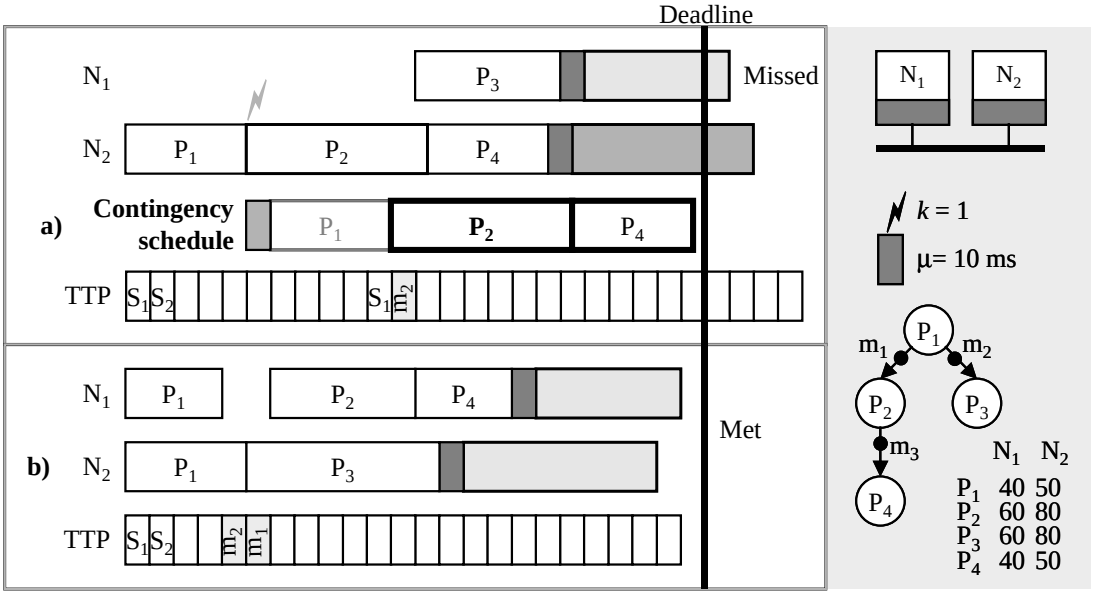


Figure 4. Combining Re-execution and Replication

data dependent on P_2, the deadline is missed in Figure 3b$_2$ if re-execution is used, and it is met when replication is used as in Figure 3a$_2$.

Note that in Figure 3b$_1$ processes P_1 and P_2 can use the same slack for re-execution. Similarly, in Figure 3b$_2$, one single slack of size $C_3 + \mu$ is enough to tolerate one fault in any of the processes. In general, re-execution slacks can be shared as long as they allow a re-execution of processes to tolerate faults.

This example shows that the particular technique to use, has to be carefully adapted to the characteristics of the application. Moreover, the best result is most likely to be obtained when both techniques are used together, some processes being re-executed, while others replicated. Let us consider the example in Figure 4, where we have an application with four processes mapped on an architecture of two nodes. In Figure 4a all processes are re-executed, and the depicted schedule is optimal for re-execution.

We use a particular type of re-execution, called transparent re-execution [10], that hides fault occurrences on a processor from other processors. On a processor N_i where a fault occurs, the scheduler has to switch to a contingency schedule that delays descendants of the faulty process. However, a fault happening on another processor, is not visible on N_i, even if the descendants of the faulty process are mapped on N_i. For example, in order to isolate node N_1 from the occurrence of a fault in P_1 on node N_2, message

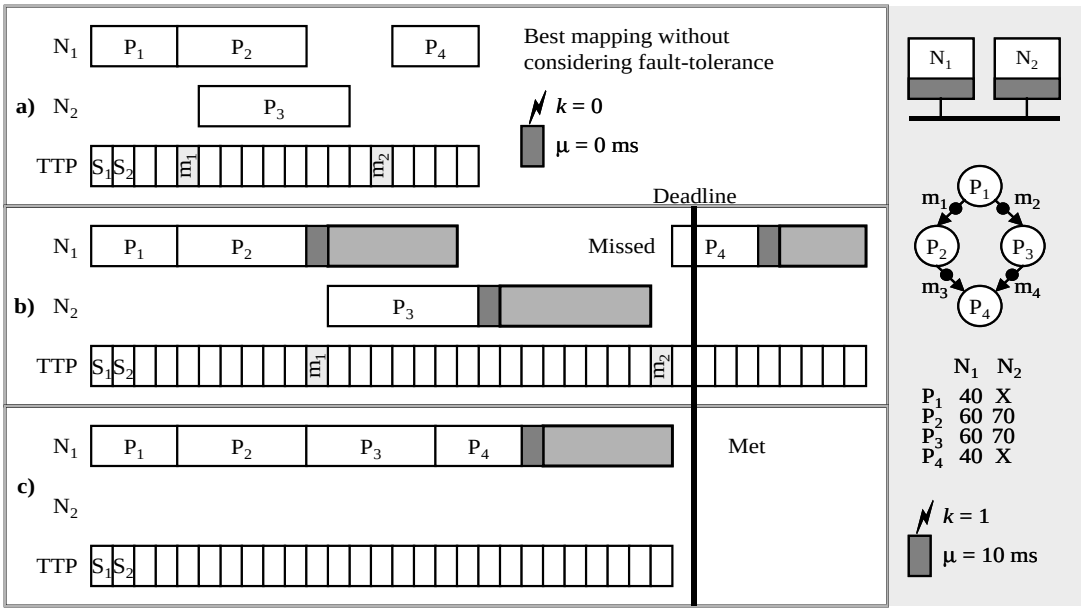


Figure 5. Mapping and Fault-Tolerance

m_2 from P_1 to P_3 cannot be transmitted at the end of P_1's execution. Message m_2 has to arrive at the destination even in the case of a fault occurring in P_1, so that P_3 can be activated on node N_2 at a fixed start time, regardless of what happens on node N_1, i.e., transparently. Consequently, m_2 can only be transmitted after a time $C_1 + \mu$ passes, at the end of the potential re-execution of P_1, depicted in grey. Message m_2 is delivered in the slot S_2 of the TDMA round corresponding to node N_2. With this setting, using re-execution will miss the deadline. Once a fault happens, the scheduler in N_2 will have to switch to a contingency schedule, depicted with thick-border rectangles.

However, combining re-execution with replication, as in Figure 4b where process P_1 is replicated, will meet the deadline. In this case, message m_2 does not have to be delayed to mask the failure of process P_1. Instead, P_2 and P_3 will have to receive m_1 and m_2, respectively, from both replicas of P_1, which will introduce a delay due to the interprocessor communication on the bus.

4.2 Mapping and Bus Access Optimization

For a distributed system, the communication infrastructure has an important impact on the mapping decisions [21]. Not only is the mapping influenced by the protocol setup, but the fault-tolerance policy assignment cannot be done separately from the mapping design task. Consider the example in Figure 5. Let us suppose that we have applied a mapping algorithm without considering the fault-tolerance aspects, and we have obtained the best possible mapping, depicted in Figure 5a. If we apply on top of this mapping a fault-tolerance technique, for example, re-execution as in Figure 5b, we miss the deadline. The re-execution has to be considered during the mapping process, and then the best mapping will be the one in Figure 5c which clusters all processes on the same processor in order to reduce the re-execution slack and the delays due to the masking of faults.

In this paper, we will consider the assignment of fault-tolerance policies at the same time with the mapping of processes to processors. However, to simplify the presentation we will not discuss the optimization of the communication channel. Such an optimization can be performed with the techniques we have proposed in [19] for non fault-tolerant systems.

5. Design Optimization Strategy

The design problem formulated in the previous section is NP complete. Our strategy is outlined in Figure 6 and has three steps:

1. In the first step (lines 1–3) we decide very quickly on an initial bus access configuration B^0, and an initial fault-tolerance policy assignment F^0 and mapping M^0. The initial bus access configuration (line 1) is determined by assigning nodes to the slots ($S_i = N_i$) and fixing the slot length to the minimal allowed value, which is equal to the length of the largest message in the application. The initial mapping and fault-tolerance policy assignment algorithm (InitialMPA line 2 in Figure 6) assigns a re-execution policy to each process in P^+ and produces a mapping for the processes in P^* that tries to balance the utilization among nodes and buses. The application is then scheduled using the ListScheduling algorithm outlined in Section 5.1. If the application is schedulable the optimization strategy stops.
2. The second step consists of a greedy heuristic GreedyMPA (line 4), discussed in Section 5.2, that aims to improve the fault-tolerance policy assignment and mapping obtained in the first step.
3. If the application is still not schedulable, we use, in the third step, a tabu search-based algorithm TabuSearchMPA presented in Section 5.2. Finally, the bus access optimization is performed.

If after these three steps the application is unschedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

```
OptimizationStrategy(A, N)
1  Step 1:B^0 = InitialBusAccess(A, N)
2          ψ^0 = InitialMPA(A, N, B^0)
3          if S^0 is schedulable then stop end if
4  Step 2:ψ= GreedyMPA(A, N, ψ^0)
5          if S is schedulable then stop end if
6  Step 3:ψ = TabuSearchMPA(A, N, ψ)
7  return ψ
end OptimizationStrategy
```

Figure 6. The General Strategy

5.1 List Scheduling

Once a fault-tolerance policy and a mapping are decided, as well as a communication configuration is fixed, the processes and messages have to be scheduled. We use a list scheduling algorithm for building the schedule tables for the processes and deriving the MEDL for messages.

Before applying list scheduling, we merge the application graphs into one single merged graph Γ, as detailed in [18], with a period equal to the LCM of all constituent graphs. List scheduling heuristics are based on priority lists from which processes are extracted in order to be scheduled at certain moments. A process P_i is placed in the ready list L if all its predecessors have been already scheduled. All ready processes from the list L are investigated, and that process P_i is selected for placement in the schedule which has the highest priority. We use the modified partial critical path priority function presented in [20]. At the same time with placing processes in the schedule, the messages are also scheduled using the ScheduleMessage function from [20]. The ListScheduling loops until the ready list L is empty.

During scheduling, re-execution slack is introduced in the schedule for the re-executed processes. The introduction of re-execution slack is discussed in [10] where the total amount of slack is reduced through slack-sharing, as depicted in Figure $3b_2$, where processes P_1 to P_3 can share the same slack for re-execution in the case of a fault.

However, the notion of "ready process" in [10] is different for us in the case of processes waiting inputs from replicas. In that case, a process can be placed in the schedule as soon as we are certain that at least one valid message has arrived from a replica. Let us consider the example in Figure 7, where P_2 is replicated. In the worst-case fault-

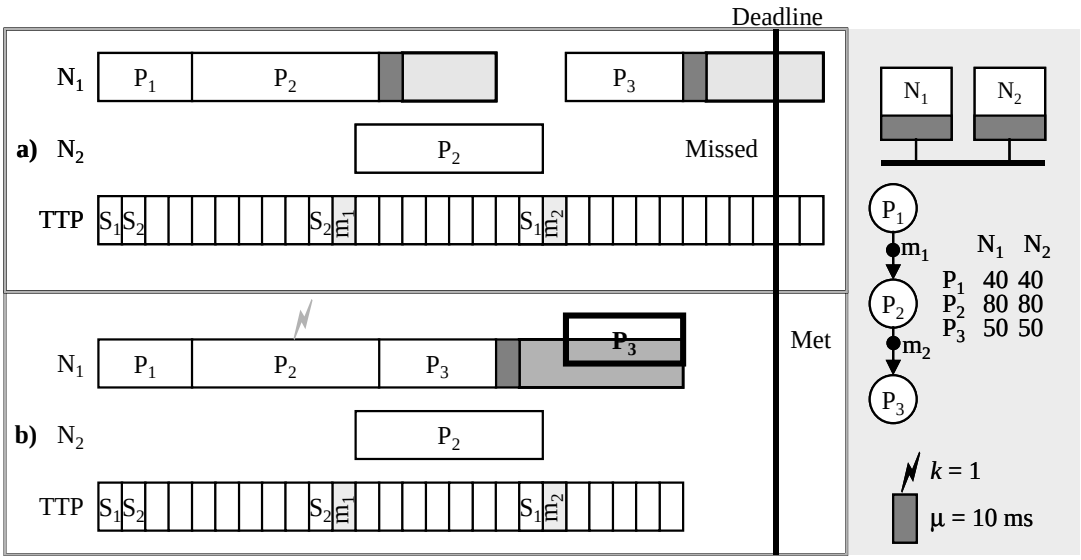


Figure 7. Scheduling Replica Descendants

scenario, P_2 on processor N_1 can fail, and thus P_3 has to receive message m_2 from the P_2 replica on processor N_2. Thus, P_3 has to be placed in the schedule as in Figure 7a. However, our scheduling algorithm will place P_3 as in Figure 7b instead, immediately following P_2 on N_1. In addition, it will create a contingency schedule for P_3 on processor N_1, as depicted in Figure 7b using a rectangle with a thicker margin. The scheduler on N_1 will switch to this schedule only in the case of an error occurring in P_2 on processor N_1. This contingency schedule has two properties. First, P_3 starts such that the arrival of m_2 from the P_2's replica on N_2 is guaranteed. Up to this point, it looks similar to the case in Figure 7a, where P_3 has been started at this time from the beginning. However, the contingency schedule has another important property: although P_3's failure is handled through re-execution, the contingency schedule will not contain any re-execution slack for P_3. That is because, according to the fault model, no more errors can happen. Thus, the deadline is met, even if any of the processes will experience a fault.

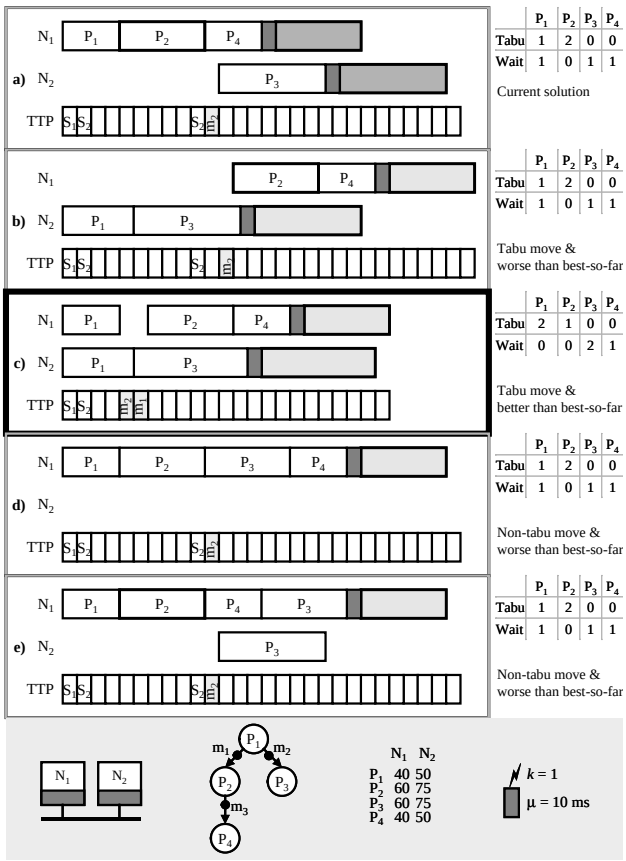


Figure 8. Moves and Tabu History

5.2 Mapping and Fault-Policy Assignment

For deciding the mapping and fault-policy assignment we use two steps, see Figure 6. One is based on a greedy heuristic, GreedyMPA. If this step fails, we use in the next step a tabu search approach, TabuSearchMPA.

Both approaches investigate in each iteration all the processes on the critical path of the merged application graph Γ, and use design transformations (moves) to change a design such that the critical path is reduced. Let us consider the example in Figure 8, where we have an application of four processes that has to tolerate one fault, mapped on an architecture of two nodes. Let us assume that the current solution is the one depicted in Figure 8a. In order to generate neighboring solutions, we perform design transformations that change the mapping of a process, and/or its fault-tolerance policy. Thus, the neighbor solutions generated starting from Figure 8a, are the solutions presented in Figure 8b–8e. Out of these, the solution is Figure 8c is the best in terms of schedule length.

The greedy approach selects in each iteration the best move found and applies it to modify the design. The disadvantage of the greedy approach is that it can "get stuck" into a local optima. To avoid this, we have implemented a tabu search algorithm, presented in Figure 9.

The tabu search takes as an input the merged application graph Γ, the architecture N and the current implementation

```
TabuSearchMPA(Γ, N, ψ)
1  -- given a merged application graph Γ and an architecture N produces a policy
2  -- assignment F and a mapping M such that Γ is fault-tolerant & schedulable
3  x^best = x^now = ψ; BestCost = ListScheduling(Γ, N, x^best) -- Initialization
4  Tabu = ∅; Wait = ∅ -- The selective history is initially empty
5  while x^best not schedulable ∧ TimeLimit not reached do
6    -- Determine the neighboring solutions considering the selective history
7    CP = CriticalPath(Γ); N^now = GenerateMoves(CP)
8    -- eliminate tabu moves if they are not better than the best-so-far
9    N^tabu = {move(P_i) | ∀ P_i ∈ CP ∧ Tabu(P_i)=0 ∧ Cost(move(P_i)) < BestCost}
10   N^non-tabu = N \ N^tabu
11   -- add diversification moves
12   N^waiting = {move(P_i) | ∀ P_i ∈ CP ∧ Wait(P_i) > |Γ|}
13   N^now = N^non-tabu ∪ N^waiting
14   -- Select a solution based on aspiration criteria
15   x^now = SelectBest(N^now);
16   x^waiting = SelectBest(N^waiting); x^non-tabu = SelectBest(N^non-tabu)
17   if Cost(x^now) < BestCost then x = x^now -- select x^now if better than best-so-far
18     else if ∃ x^waiting then x = x^waiting -- otherwise diversify
19     else x = x^non-tabu -- if no better and no diversification, select best non-tabu
20   end if
21   -- Perform selected move
22   PerformMove(x); Cost = ListScheduling(Γ, N, x)
23   -- Update the best-so-far solution and the selective history tables
24   If Cost < BestCost then x^best = x; BestCost = Cost end if
25   Update(Tabu); Update(Wait)
26 end while
27 return x^best
end TabuSearchMPA
```

Figure 9. The Tabu Search Algorithm

ψ, and produces a schedulable and fault-tolerant implementation x^{best}. The tabu search is based on a neighborhood search technique, and thus in each iteration it generates the set of moves N^{now} that can be reached from the current solution x^{now} (line 7 in Figure 9). In our implementation, we only consider changing the mapping or fault-tolerance policy of the processes on the critical path, denoted with *CP* in Figure 9. We define the critical path as the path through the merged graph Γ which corresponds to the longest delay in the schedule table. For example, in Figure 8a, the critical path is formed from P_1, m_2 and P_3.

The key feature of a tabu search is that the neighborhood solutions are modified based on a selective history of the states encountered during the search. The selective history is implemented in our case through the use of two tables, *Tabu* and *Wait*. Each process has an entry in this tables. If $Tabu(P_i)$ is non-zero, it means that the process is "tabu", i.e., should not be selected for generating moves, while if $Wait(P_i)$ is greater than the number of processes in the graph, |Γ|, the process has waited a long time and should be selected for diversification. Thus, lines 9 and 10 of the algorithm, a move will be removed from the neighborhood solutions if it is tabu. However, tabu moves are also accepted if they are better than the best-so-far solution (line 10). In line 12 the search is diversified with moves which have waited a long time without being selected.

In lines 14–20 we select the best one out of these solutions. We prefer a solution that is better than the best-so-far x^{best} (line 17). If such a solution does not exist, then we choose to diversify. If there are no diversification moves, we simply choose the best solution found in this iteration, even if it is not better than x^{best}. Finally, the algorithm updates the best-so-far solution, and the selective history tables *Tabu* and *Wait*. The algorithm ends when a schedulable solutions has been found, or an imposed time-limit has been reached.

Figure 8 illustrates how the algorithm works. Let us consider that the current solution x^{now} is the one presented in Figure 8a, with the corresponding selective history presented to its right, and the best-so-far solution x^{best} is the one in Figure 4a. The generated solutions are presented in Figure 8b–8e. The solution (b) is removed from the set of considered solutions because it is tabu, and it is not better than x^{best}. Thus, solutions (c)–(e) are evaluated in the current iteration. Out of these, the solution in Figure 8c is selected, because although it is tabu, it is better than x^{best}. The table is updated as depicted to the right of Figure 8c in bold, and the iterations continue with solution (c) as the current solution.

6. Experimental Results

For the evaluation of our algorithms we used applications of 20, 40, 60, 80, and 100 processes (all unmapped and with no fault-tolerance policy assigned) implemented on architectures consisting of 2, 3, 4, 5, and 6 nodes, respectively. We have varied the number of faults depending on the architecture size, considering 3, 4, 5, 6, and 7 faults for each architecture dimension, respectively. The duration μ of a fault has been set to 5 ms. Fifteen examples were randomly generated for each application dimension, thus a total of 75 applications were used for experimental

Table 1. Overheads of MXR compared to NFT

(a) Application size					(b) Number of faults 60 procs., μ=5				(c) μ 20 procs., *k*=3			
procs.	*k*	% max	% avg.	% min	*k*	% max	% avg.	% min	μ	% max	% avg.	% min
20	**3**	98.36	70.67	48.87	**2**	52.44	32.72	19.52	**1**	78.69	57.26	34.29
40	**4**	116.77	84.78	47.30	**4**	110.22	76.81	46.67	**5**	95.90	70.67	48.87
60	**5**	142.63	99.59	51.90	**6**	162.09	118.58	81.69	**10**	122.95	89.24	67.58
80	**6**	177.95	120.55	90.70	**8**	250.55	174.07	117.84	**15**	132.79	107.26	75.82
100	**7**	215.83	149.47	100.37	**10**	292.11	219.79	154.93	**20**	149.01	125.18	95.60

evaluation. We generated both graphs with random structure and graphs based on more regular structures like trees and groups of chains. Execution times and message lengths were assigned randomly using both uniform and exponential distribution within the 10 to 100 ms, and 1 to 4 bytes ranges, respectively. The experiments were done on Sun Fire V250 computers.

We were first interested to evaluate the proposed optimization strategy in terms of overheads introduced due to fault-tolerance. Hence, we have implemented each application, on its corresponding architecture, using the OptimizationStrategy (MXR) strategy from Figure 6. In order to evaluate MXR, we have derived a reference non-fault tolerant implementation, NFT. The NFT approach is an optimized implementation similar to MXR, but we have removed the moves that decide the fault-tolerance policy assignment. To the NFT implementation thus obtained, we would like to add fault-tolerance with as little as possible overhead, and without adding any extra hardware resources. For these experiments, we have derived the shortest schedule within an imposed time limit: 10 minutes for 20 processes, 20 for 40, 1 hour for 60, 2 hours and 20 min. for 80 and 5 hours and 30 min. for 100 processes.

The first results are presented in Table 1a, where we have two columns, the first column presents the average overheads introduced by MXR compared to NFT, while the second column presents the minimum overhead. Let δ_{MXR} and δ_{NFT} be the schedule lengths obtained using MXR and NFT, respectively. The overhead is defined as $100 \times (\delta_{MXR} - \delta_{NFT}) / \delta_{NFT}$. We can see that the overheads due to fault-tolerance grow with the application size. MXR approach can offer fault-tolerance within the constraints of the architecture at an average overhead of approximately 100%. However, even for applications of 60 processes, there are cases where the overhead is as low as 52%.

We were also interested to evaluate our MXR approach in the case the number of faults and their length varies. We have considered applications with 60 processes mapped on four processors, and we have varied the number *k* of faults

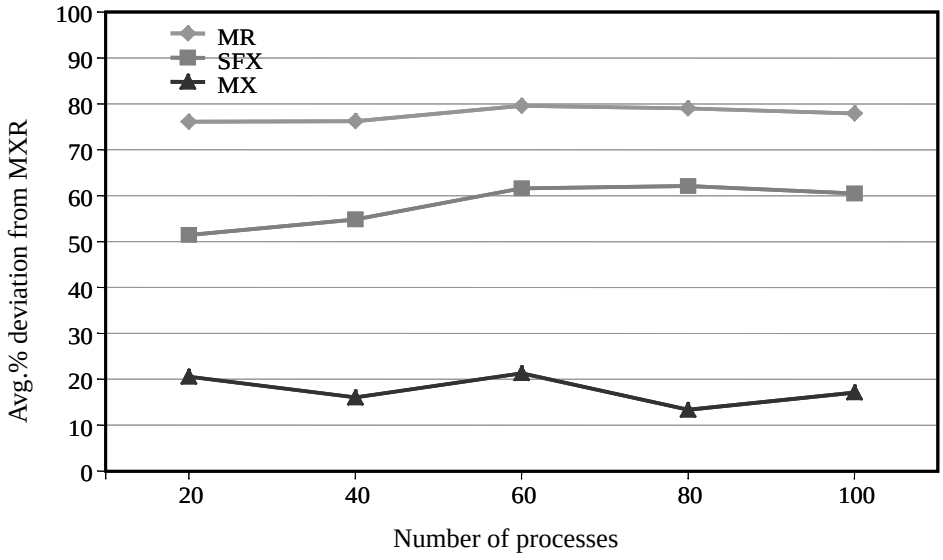


Figure 10. Comparison of MXR with MX, MR and SFX

from 2, 4, 6, 8, to 10, using a constant μ = 5 ms. Table 1b shows that the overheads increase substantially as the number of faults that have to be tolerated increase. This is to be expected, since we need more replicas and/or re-executions if there are more faults. Similarly, we have kept the number of faults constant to 3, and varied μ: 1, 5, 10 15 and 20 ms, for 20 processes and two processors. We can observe in Table 1c that the overhead also increases with the increase in fault duration. However, the increase due to the fault duration is significantly lower compared to the increase due to the number of faults.

As a second set of experiments, we were interested to evaluate the quality of our MXR optimization approach. Thus, together with the MXR approach we have also evaluated two extreme approaches: MX that considers only re-execution, and MR which relies only on replication for tolerating faults. MX and MR use the same optimization approach as MRX, but besides the mapping moves, they consider assigning only re-execution or replication, respectively. In Figure 10 we present the average percentage deviations of the MX and MR from MXR in terms of overhead. We can see that by optimizing the combination of re-execution and replication, MXR performs much better compared to MX and MR. On average, MXR is 77% and 17.6% better than MR and MX, respectively. There are also situations, for graphs with 60 processes, for example, where MXR is able to reduce the overhead with up to 40% compared to MX, and up to 90% compared to MR. This shows that considering re-execution at the same time with replication can lead to significant improvements.

In Figure 10 we have also presented a straightforward strategy SFX, which first derives a mapping without fault-tolerance considerations (using MXR without fault-tolerance moves) and then applies re-execution. This is a solution that can be obtained by a designer without the help with our fault-tolerance optimization tools. We can see that the overheads thus obtained are very large compared to MXR, up to 77% more on average. This shows that the optimization of the fault-tolerance policy assignment has to be addressed at the same time with the mapping of functionality. In Figure 10 we also see that replication (MR) is worst than even the straightforward re-execution (SFX). However, by carefully optimizing the usage of replication alongside re-execution (MXR), we are able to obtain results that are significantly better than re-execution only (MX).

Finally, we considered a real-life example implementing a vehicle cruise controller (CC). The process graph that models the CC has 32 processes, and is described in [18]. The CC was mapped on an architecture consisting of three nodes: Electronic Throttle Module (ETM), Anti-lock Breaking System (ABS) and Transmission Control Module (TCM). We have considered a deadline of 250 ms, k = 2 and μ = 2 ms.

In this setting, the MRR produced a schedulable fault-tolerant implementation with a worst-case system delay of 229 ms, and with an overhead compared to NFT of 65%. If only one policy is used for fault-tolerance, as in the case of MX and MR, with 253 and 301 ms, respectively, the deadline is missed.

7. Conclusions

In this paper we have addressed the optimization of distributed embedded systems for fault-tolerance hard real-time applications. The processes are scheduled with static cyclic scheduling, while for the message transmission we use the TTP. We have employed two fault-tolerance techniques for tolerating faults: re-execution, which provides time-redundancy, and active replication, which provides space-redundancy.

We have implemented a tabu search-based optimization approach that decides the mapping of processes to the architecture and the assignment of fault-tolerance policies to processes. Our main contribution is that we have considered the interplay of fault-tolerance techniques for reducing the overhead due to fault-tolerance. As our experiments have shown, by carefully optimizing the system implementation we are able to provide fault-tolerance under limited resources.

References

[1] A. Bertossi, L. Mancini, “Scheduling Algorithms for Fault-Tolerance in Hard-Real Time Systems”, *Real Time Systems Journal*, 7(3), 229–256, 1994.

[2] A. Burns et al., “Feasibility Analysis for Fault-Tolerant Real-Time Task Sets”, *Proceedings of Eighth Euromicro Workshop on Real-Time Systems*, 29–33, 1996.

[3] P. Chevochot, I. Puaut, “Scheduling Fault-Tolerant Distributed Hard-Real Time Tasks Independently of the Replication Strategies”, *Sixth International Conference on Real-Time Computing Systems and Applications*, 356–363, 1999.

[4] V. Claeson, S. Poldena, J. Söderberg, “The XBW Model for Dependable Real-Time Systems”, *Proceedings of International Conference on Parallel and Distributed Systems*, 1998.

[5] C. Dima et al, “Off-line Real-Time Fault-Tolerant Scheduling”, *Proceedings of Ninth Euromicro Workshop on Parallel and Distributed Processing*, 410–417, 2001.

[6] G. Fohler, “Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems”, *Proceedings of 16th IEEE Real-Time Systems Symposium*, 152–161, 1995.

[7] G. Fohler, “Adaptive Fault-Tolerance with Statically Scheduled Real-Time Systems”, *Proceedings of Ninth Euromicro Real-Time Systems Workshop*, 161–167, 1997.

[8] C. C. Han, K. G. Shin, J. Wu, “A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults”, *IEEE Transactions on Computers*, 52(3), 362–372, 2003.

[9] K. Hoyme, K. Driscoll, “SAFEbus”, *IEEE Aerospace and Electronic Systems Magazine*, 8(3), 34–39, 1992.

[10]N. Kandasamy, J. P. Hayes, B. T. Murray, “Transparent Recovery from Intermittent Faults in Time-Triggered

Distributed Systems", *IEEE Transactions on Computers*, 52(2), 113–125, 2003.

[11]N. Kandasamy, J. P. Hayes B.T. Murray "Dependable Communication Synthesis for Distributed Embedded Systems", *Procedeeng of 22nd International Conference on Computer Safety, Reliability, and Security, Lecture Notes in Computer Science 2788, Springer-Verlag*, 275–288, 2003.

[12]H. Kopetz, "Real-Time Systems–Design Principles for Distributed Embedded Applications", *Kluwer Academic Publishers*, 1997.

[13]H. Kopets et al., "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", *IEEE Micro*, 9(1), 25–40, 1989.

[14]H. Kopetz, Günter Bauer, "The Time-Triggered Architecture", *Proceedings of the IEEE*, 91(1), 112–126, 2003.

[15]C. Pinello, L. P. Carloni, A. L. Sangiovanni-Vincentelli, "Fault-Tolerant Deployment of Embedded Software for Cost-Sensitive Real-Time Feedback-Control Applications", *Design, Automation and Test in Europe Conference and Exhibition*, 1164–1169, 2004.

[16]S. Poldena, "Fault Tolerant Systems–The Problem of Replica Determinism", *Kluwer Academic Publishers*, 1996.

[17]Y. Zhang, K. Chakrabarty, "Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems", *Design, Automation and Test in Europe Conference and Exhibition*, 918–923, 2003.

[18]P. Pop, "Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems", *Ph. D. Thesis No. 833, Dept. of Computer and Information Science, Linköping University*, 2003.

[19]P. Pop, P. Eles, Z. Peng, "Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems", *Design, Automation and Test in Europe Conference and Exhibition*, pp. 184-189, 2003.

[20]P. Eles et al., "Scheduling with Bus Access Optimization for Distributed Embedded Systems", *IEEE Transactions on VLSI Systems*, 8(5), 472-491, 2000.

[21]P. Pop et al., "Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications", *Design, Automation and Test in Europe Conference and Exhibition*, pp. 1028-1033, 2004.

Global Block RAM Allocation and Accesses for FPGA Implementation of Real-Time Video Processing Systems

Najeem Lawal
Electronic Design Division, Department of Information Technology & Media,
Mid Sweden University, SE-851 70, Sundsvall, Sweden
Najeem.Lawal@miun.se

Abstract

FPGA offers the potential to become a reliable, and high-performance reconfigurable platform for the implementation of real-time video processing systems. To utilize the full processing power of FPGA for video processing applications, optimization of memory allocation architecture and memory accesses are important issues. This paper presents an algorithm for the allocation of on-chip FPGA Block RAMs for the implementation of Real-Time Video Processing Systems and two approaches, base pointer approach and distributed pointer approach, to implement accesses to on-chip FPGA Block RAMs. The effectiveness of the algorithm is shown through the implementation of realistic image processing systems. The algorithm, which is based on a heuristic, seeks the most cost effective way of allocating memory objects to the FPGA Block RAMs. The experimental results show that the algorithm generates results which are close to the theoretical optimum for most design cases. A comparison of the two memory accessing approaches on realistic image processing systems design cases also is presented. Results show that compared to the base pointer approach, the distributed pointer approach increases the potential processing power of FPGA, as a reconfigurable platform for video processing systems.

1 Introduction

In a Real-Time Video Processing System (RTVPS), the operations performed on each pixel are often neighbourhood oriented [1]. A neighbourhood of pixels is an operation window about a point in the image and is usually a square or rectangular sub-image about that point. These pixels act as input data on which operations are performed to yield a pixel that corresponds to the central neighbourhood pixel in the output image. The neighbourhood is formed for each pixel in the input image to produce an output image. One consequence of this is that a large amount of data is required to be buffered depending on the size of the video frame and the operation window. The data flow dependencies require data to be stored in buffers, where each buffer normally corresponds to a row in the video frame. The size of each element in this buffer depends on the dynamic range of the video signal. In addition, the dimensions of the process window affect the number of buffers required for the process operation. For a 5x5 window, four line buffers are required while two line buffers are required for a 3x3 window.

Due to its high processing capability, FPGA is an acceptable alternative for implementing high speed, real-time, computation-intensive operations, as in the case of RTVPS [2]. Also, the on-chip memories of an FPGA allow frequently accessed data to be stored close to the data path, thus eliminating the overheads associated with data fetches to external memory.

The available FPGAs on-chip memories are limited and organised as Block RAMs and distributed RAMs. Distributed RAMs are built from the logic resources and are ideal for register files closely integrated with logic. The Block RAMs are more suited to larger on-chip memory storage requirements, such as buffers and caches. For Xilinx FPGA, each Block RAM is a configurable, synchronous block of memory [3, 4]. Additionally, it is also possible to configure each Block RAM as a single- or dual-port memory. For the dual port configuration, the two data ports permit independent synchronous read/write access to the common Block RAM. In addition, the data path widths at each port are independent of each other. Examples of allowable data path widths at each data port for Spartan 3 are 1, 2, 4, 8 (9), 16 (18) and 32 (36) [4]. The values shown in brackets are available when parity bits are used for data storage.

2 Related Works

Attempts at optimising energy consumption, total area and availability of data from the memory subsystems of FPGA and DSP have attracted many research efforts [5 - 7]. This could be explained by the work of Balasa et al. who state that the area occupied and the power consumed by the memory subsystems are up to ten times larger than that of the data-path [8]. Memory accesses are a major contributor to the power consumption for RTVPS. In addition, latency in memory accesses affects the throughput of the RTVPS. Effective optimisation can be achieved through efficient memory architecture and addressing procedure.

There are several approaches available for the development of methods for the efficient handling of data storage and memory transfers. Data Transfer and Storage Exploration (DTSE) is a methodology that supports analysis and optimization of data dominated applications [9]. ATOMIUM is a tool that supports DTSE and can take a DSP algorithm all the way from high-level specification to customized hardware and processor level implementations. The main focus throughout all the design steps is towards memory usage and transactions. In DTSE, the search within a complete design space for an optimal implementation is performed in well-defined steps. The allocation of these steps in the search trajectory is based on heuristics.

Weinhardt and Luk present a technique that optimally allocates memory objects to a set of background memory banks using Integer Linear Programming (ILP) [10]. This technique achieves optimisation by parallelisation of data accesses through pipelining. The optimisation technique cannot distribute a single memory object over several memory banks. Thus, each single ported memory bank is required to be sufficiently large to hold each memory object. Consequently, all accesses to each memory object must be serialised.

The MeSA algorithm is based on the clustering of array variables to determine the memory configuration that will result in the minimum total memory area [11]. The number of memory modules, the size of each module, the number of ports for each module and the grouping of a set of input array variables are computed. The number of ports is balanced for serialized memory accesses within a control and data flow graph. This algorithm cannot however be considered for implementing RTVPS on FPGA. This is because large array variables cannot be distributed among a set of memory modules.

A general approach to FPGA memory allocation and assignment is presented by Gokhale et al. [12] This approach starts from C code, for which the presented method allocates both external and internal memories. Automatic partitioning of a single array among different memories is however not covered by this work.

Jaewon et al. propose a schedule based memory allocation, which allows non-uniform access speeds among memory ports [13]. This heuristic considers latency constraints at the scheduling stage and exploits DRAM page mode accesses at both the array clustering and assignment phases. Gebotys considers scheduling both the register and memory accesses among different control steps [14]. The optimization goal is low energy and is achieved using a network flow model.

None of the allocation algorithms mentioned in this section can fully utilize the FPGA architectures in combination with RTVPS [17]. Either it is not possible to use a fine-grained set of memories, or the algorithms are too complicated as they include the scheduling of serialized accesses. None of the mentioned allocation algorithms can utilize the configurable port data widths supported by the FPGA architectures. This is our motivation for presenting a new memory architecture- and application specific allocation algorithm that can fully utilize the combination of FPGAs and RTVPS.

The work presented by Doggett el al. for memory accessing is optimal where large numbers of memory banks are used as is typical in volume rendering in medical applications [5]. The address generation scheme by Grant, el al. is an efficient option for accessing data with addresses within the power range of two [6]. The scheme uses a register and optionally an offset, to the specify memory read/write addresses.

The memory exploration algorithm in [7] implements memory allocation and array-mapping to RAMs through tight links to the scheduling effect and non-uniform access speeds among the RAM ports to achieve near optimal memory area and efficient energy requirement. The algorithm is however complex and the execution time may slow down hardware design and performance. Moreover the exploration targets SRAM and DRAM as opposed to the on-chip FPGA Block RAMs, which are the focus of this paper. Many efficient, often heuristics based, memory optimisation algorithms have been developed similar to those in [9, 10], however, most of these are tailored to be efficient using DSP. Thus a memory access methodology

is required for the FPGA platform which takes advantage of the global memory allocation architecture proposed by O'Nils et al. for RTVPS implementation [17].

Based on the global memory allocation architecture, an allocation algorithm has been developed and implemented to maximize memory usage while minimising the read/write accesses. This algorithm is presented in this paper. This paper also presents an algorithm that generates addresses for the results of the allocation algorithm for the global memory objects proposed in [17] in order to create a hardware description for accessing the allocated memory. Two different approaches and a comparison of these are presented in this paper. This work is intended to be used in the IMEM library of design tools for which the design flow of the memory sub-component is shown in Figure 1 [18]. The result of the Memory Allocation, stage which is information about organisation and allocation of the RTVPS memory objects, serves as the input to the memory access and address generation stage. The contribution of this work is the generation of VHDL code for high performance hardware implementation of memory allocation to FPGA Block RAM and their accesses in RTVPS. This work is also applicable to memory access formulation in general where the target of the design tool is the FPGA platform.

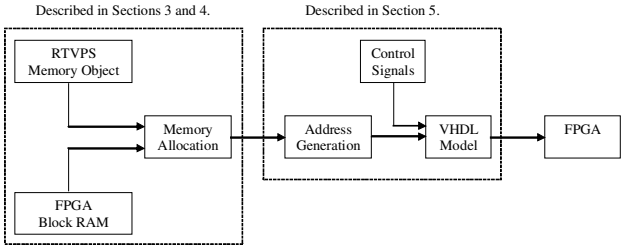


Figure 1. Proposed Design Flow.

3 Architectural Assumptions

For neighbourhood oriented operations in an RTVPS, line buffers (memory objects) are required to store the data of the neighbourhood pixels. Figure 2 shows a 3x3 pixel neighbourhood for which an image processing operator calculates an output value for the central pixel. In a streamed hardware implementation only one operator can use these memory objects and, all the memory objects are used simultaneously in the RTVPS. It is assumed that the memory objects can be grouped together to form a global memory object for the operator. This grouping can be achieved through:

$$W_{Ri} = n_{lines} \times w_p \tag{1}$$

where W_{Ri} is the width of the Global Memory Object (GMO) for the operator, n_{lines} is the number of required line buffers for an operator and w_p is the bit width representing a pixel. The length of the global memory object is equal to those of the memory objects that formed it, i.e. the image width [17].

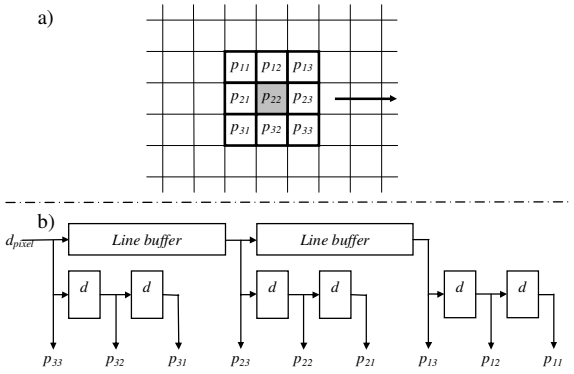


Figure 2. Basic architecture for the implementation of line buffers in neighbourhood oriented image processing operations. Part a) shows an example of a 3×3 neighbourhood and b) its implementation.

This architecture is preferable to that proposed by Norell et al. in which each line buffer (memory module) is mapped directly to memory [19]. This preference is because GMOs require a minimal number of required memory entities in comparison to direct mapping architecture. Consequently, the number of memory accesses for an RTVPS operation is minimal for a global memory object.

4 Memory Allocation

A heuristics-based algorithm for efficient allocation of the GMOs, based on the architecture stated above has been developed and implemented. The algorithm creates the GMOs based on Eq. (1) by grouping memory objects required by an operator together in an RTVPS. It partitions the GMOs to ensure that their widths conform to those specified by the FPGA. The algorithm takes advantage of the dual port capabilities of the Block RAMs to achieve optimal allocations and the possibility of allocating a GMO to as many Block RAMs as required. this allocation algorithm is presented in this section.

For every Block RAM available on the FPGA, the algorithm attempts to allocate it a GMO. Final allocation decisions requiring as few Block RAMs as possible are made, based on the allocations offering the least amount of unused memory space on the allocated Block RAMs. Figure 3 depicts a model of the allocation.

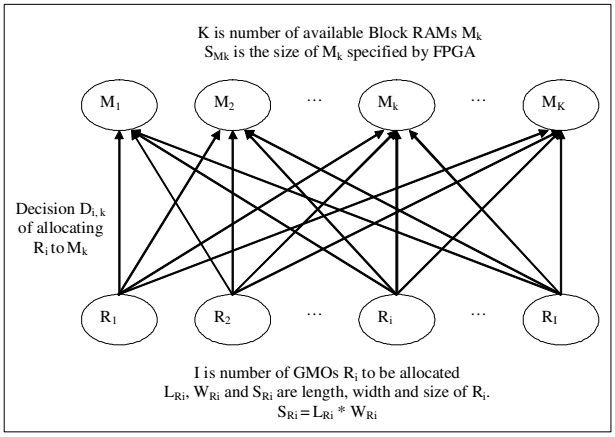


Figure 3. Allocation model.

4.1 Definitions

To find the optimal use of the Block RAM, the allocation algorithm must observe a few definitions and constraints. These are listed as follows:

(i) M is the set of all available Block RAM M_k and K is the number of Block RAMs.

$$M = \{M_k | k = 1,2,...,K\} \tag{2}$$

(ii) S_{Mk} is the size of the Block RAM M_k and is specified by the FPGA. For example, in Xilinx Spartan 2E FPGA S_{Mk} is 4096 bits [2]. The memory objects allocated to the Block RAM determine the length L_{Mk} and width W_{Mk} of M_k.

(iii) W_c is the set of all possible datapath widths W_n for Block RAMs on the FPGA. For example, 1, 2, 4, 8, and 16 are allowed on Xilinx Spartan 2E FPGA [2].

$$W_c = \{W_n | n = 1,2,...,N\} \tag{3}$$

(iv) R is the set of all memory objects R_i to be allocated and I is the number of memory objects.

$$R = \{R_i | i = 1,2,...,I\} \tag{4}$$

(v) The size S_{Ri} of memory object R_i is defined as product of the length L_{Ri} and data width W_{Ri} of the memory object R_i.

$$S_{Ri} = L_{Ri} \times W_{Ri} \tag{5}$$

(vi) Each global memory object is characterised by a quadruple of op_id_{Ri}, L_{Ri}, W_{Ri} and x_{Ri}.

$$R_i(op\_id_{Ri}, L_{Ri}, W_{Ri}, x_{Ri}) \tag{6}$$

where op_id_{Ri} is an identifier for the operator where the memory objects making up the global memory object R_i are defined and x_{Ri} is the segment in which a memory object is located on the global memory object after partitioning into units of allowable data widths in W_c.

(vii) If W_{Ri} is not a member of W_c, R_i is partitioned into r_j partitions such that the width, w_R, of each partition is a member of W_c.

$$R_i = \left\{ r_i(op\_id_{Ri}, L_{Ri}, w_{Ri}, x_{Ri}) \middle| w_{R_i} \in W_c, j = 1,2,...,J \right\} \tag{7}$$

(viii) Memory object R_i may be allocated to as many Block RAMs as required.

$$\sum_{k=1}^{K} L_{i,k} \times W_{Ri} \leq S_{Ri} \tag{8}$$

where $L_{i,k}$ is the part of length L_{Ri} allocated at $M_{k.}$

(ix) Block RAM only supports a maximum of two data ports.

(x) $D_{i,k}$ is the decision to allocate some/ all of the memory objects R_i at M_k.

$$\sum_{i=1}^{L} D_{i,k} \leq 2 \tag{9}$$

(xi) For all R_i in R and M_k in M that form part of the $D_{i,k}$, the sum of the allocations may not be more than the size of the Block RAM.

$$\sum_{i=i}^{I} L_{i,k} \times W_{Ri} \leq S_{Mk} \tag{10}$$

4.2 Proposed algorithm

The proposed allocation algorithm is presented in Figure 4. In step 1, the algorithm creates global memory objects according to "Eq. (1)". In step 2, the algorithm ensures that they conform to the allowable port width configuration according to definition vii. This step is captured in a procedure, *confugre_global_memory_objects(R)*, presented below the algorithm in Figure 4. In steps 3 through to 10, the global memory objects are allocated to the Block RAMs according to definitions viii to xi. In steps 11 through to 20, the algorithm finds the optimal use of unallocated memory space in the Block RAM through the second port. This allocation is also in accordance with definitions viii to xi. Steps 5 and 14 handle the partitioning of the global memory objects with respect to length by allocating part of the length of the memory object to the Block RAM until the memory object has been completely allocated. In steps 7 to 9 and 15 to 17, the algorithm estimates the amount of the memory object possible for allocation to the available space on a Block RAM. This amount is used to update the memory object and the Block RAM if the allocation decision is made. In steps 18 to 20, the algorithm finds the memory object which, when allocated to the remaining space on the current Block RAM through port B, yields the optimal use of the Block RAM. The optimal allocation is that for which the unused memory space is minimum, preferably zero.

The Proposed Allocation Algorithm

```
Algorithm:  Memory Allocation(R, M)
Parameters: R[R1 … RI] set of I memory objects;
            M[M1 … MK] set of K Block RAMs;
Return:     M[M1 … MK] set of K Allocated Block RAMs;

{
1. create global memory object;
2. R := configure_global_memory_objects(R);
3. for Mk := M1 upto MK
4. { for Ri := R1 upto RI
5.   { determine length of Ri to be allocated;
6.     determine port on Mk for allocation;
7.     Allocate Ri to Mk;
8.     update Mk;
9.     update Ri;
10.    if Mk has been completely used
       { take next Mk;
       }
11.    else
12.    { if no_of_ports on Mk = 1
13.      { pair(Ri,Mk.unused) best_alloc;
14.        flag := TRUE;
15.        for Rj := Ri upto RI
16.        { determine length of Rj to be allocated
17.          temporarily Allocate Rj to Mk;
18.          temporarily update Mk;
17.          temporarily update Rj;
19.          if Mk is completely used
             { Allocate Rj to Mk;
               flag = FALSE;
               take next Mk;
             }
20.          pair(Rj,Mk.unused) temp_alloc;
21.          if temp_alloc.second < best_alloc.second
             { best_alloc := temp_alloc;
             }
           }
22.        if flag = TRUE
           { Ri := best_alloc.first;
             Allocate Ri to Mk;
             update Mk;
             update Ri;
           }
         }
       }
     }
   }
}


Procedure:  configure_global_memory_objects(R)
Parameters: R[R1 … RI] set of I memory objects;
Return:     R[R1 … RI] set of I memory objects;

{
1.  create new set of memory objects New_R;
2.  for Ri := R1 upto RI
3.  { width := Ri.width;
4.    if width ∉ Wc
5.    { segment_id := 1;
6.      foreach Wi in Wc
7.      { if width ≥ Wi
8.        { count_max := width / Wi; // integer division
9.          width := width - (Wi × count_max);
10.         for count := 1 upto count_max
11.         { Mem_Obj temp(Wi, Ri.length, Ri.operator_id);
12.           temp.set_segment(segment_id);
13.           add temp to new_R;
14.           segment_id := segment_id + 1;
            }
          }
        }
      }
15.   else
16.   { add Ri to new_R;
      }
    }
17. return new_R;
}
```

Figure 4. The proposed allocation algorithm.

The procedure for configuring the width of the global memory objects, *confugre_global_memory_objects(R)*, is based on definitions (iii) and (vii). In step 1 of the procedure, a container for the set of global memory objects is created. In this procedure, as the global memory objects are configured they are placed in this container. The container is returned in step 17 as the output of the procedure. As the procedure loops through the set of global memory objects in step 2, the width of each global memory object, $W_{Ri,}$ is obtained in step 3 and compared in step 4 with W_c. If W_{Ri} is not supported by the FPGA, the segment identifier is created in step 5. In steps 6 to 14, W_c is looped through and its members, W_n, are compared with the W_{Ri}. This comparison starts from the largest W_n down to the smallest. An appropriate number of times by which W_{Ri} is greater than W_n is used in creating segments according to definition vii. W_{Ri} is updated and reused until it is reduced to zero. If the FPGA supports W_{Ri}, in steps 15 and 16, the object is left un-partitioned and placed in the returned container.

To illustrate the function of the algorithm, if an operator in an RTVPS requires a neighbourhood of 5x5 window with 12-bit gray scale and 640 by 480 frame size as the input video stream this would result in four memory objects each of length L (=640) and width 12 being created. The memory objects would be combined to create a GMO R_i of width 48. Figure 5 depicts this illustration and *op_id* represents the operator requiring the GMO.

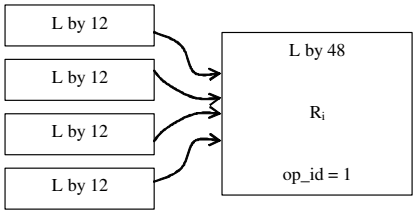


Figure 5. Creating a GMO.

The partitioning and allocation of GMOs are potentially complicated tasks depending on the length and width of GMOs in RTVPS applications and the Block RAM size. Thus an efficient technique for tracking the position of the GMO to be accessed is required. If the GMO in Figure 5 were to be allocated on a Xilinx Spartan 3 FPGA, it would be partitioned into two segments, of widths 32 and 16, since it would be not possible to have a data path width of 48 on a Xilinx Spartan 3 FPGA. In addition, since each Block RAM is 16KBit (excluding the parity feature), the first segment, of width 32, would require 2 Block RAMs, thus creating two partitions. The second segment would require a single partition on a Block RAM. Figure 6 illustrates the organisation of these partitions and their allocations on two Block RAMs. Hence organising a GMO into many segments and partitions for allocation purposes would be usual encounters in RTVPS applications. Implementation of accesses to these GMOs irrespective of their organisation, and the Block RAM constraints is presented in the next section.

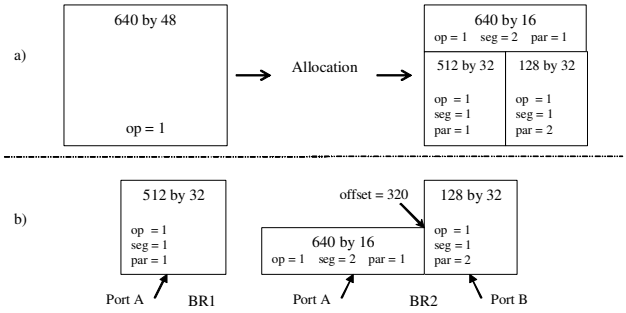


Figure 6. Organisation and Allocation of a GMO.

5 Address Generation

This work incorporates the generation of addresses for each GMO element. The allocation software ensures that each Block RAM data object knows not only the width and length of the GMO segment allocated to it, and the port used in allocation, but also the hierarchy of its segment in the GMO. In

addition, each partition stores information about the Block RAM to which it is allocated, the port of allocation and its start address on the Block RAM, the GMO and segment to which it belongs.

The advantage of sequential accesses to memory for RTVPS applications can lead to improved memory performance by using pointers that increase incrementally when there are valid pixel values. Using the GMO architecture further reduces the number of such pointers to one per RTVPS operator. The pointers may be implemented by using a single register for each GMO, further referred to as base pointer, or by using a register for each partition in a GMO, further referred to as distributed pointers.

To this end, the results of the memory allocation stage in Figure 1 are imported into the address generation module. From these allocation results, which consist basically of information about the partitions, GMOs are reconstructed, and address spans for each partition in a Block RAM are generated. The start and end addresses of each partition are calculated. Offsets are considered where dual ports are used for the allocation on Block RAMs for different partitions in order to avoid memory overlap. The generated addresses are used to determine the location of each GMO element. The descriptions for accessing the GMO elements using two approaches, namely the base pointer approach and the distributed pointer approach are presented as follows:

5.1 Base Pointer Approach

In this approach, a single pointer is used to track the location of the element to be accessed in the GMO. The pointer starts at zero and increases to one less than the length of the GMO and then resets to zero. Since the memory accesses are clocked, the value of the pointer increases with clocked access to the Block RAM when there are valid data. Address spans for each partition of the GMO are used to determine the relevant Block RAM relating to the element accessed, depending on the value of the pointer. Hence, only the relevant Block RAMs are enabled while the other related Block RAMs are disabled. Figure 7a depicts this approach for a simplified case in which a GMO consists of a single segment with two partitions.

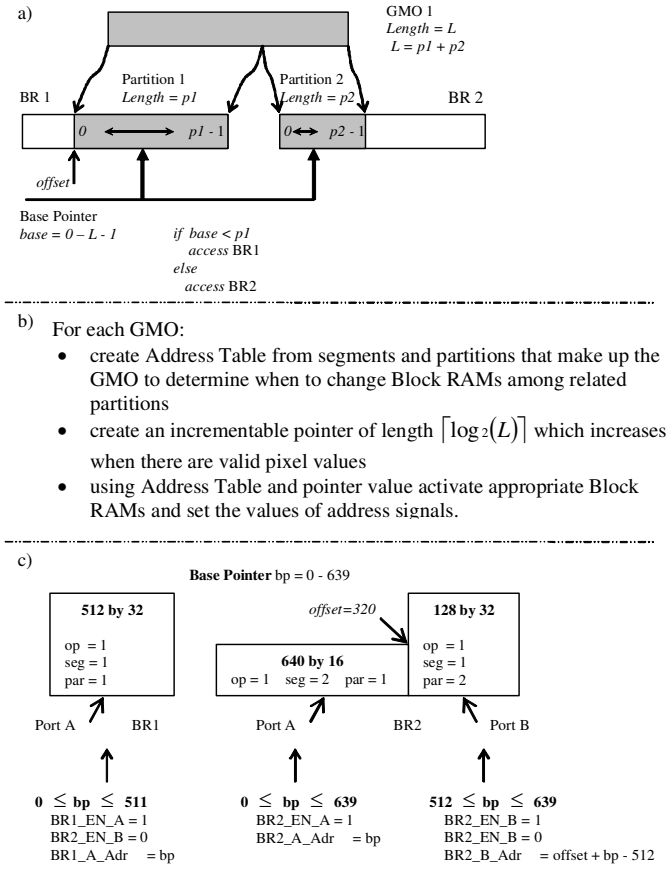


Figure 7. Base Pointer Approach.

In the figure, partitions *p1* and *p2* are allocated to Block RAMs *BR1* and *BR2*. In the figure, *BR1_EN_A*, *BR2_EN_A* and *BR2_EN_B* represent the enable signals on port A of *BR1,* port A of *BR2*, and port B of *BR2* respectively. Likewise, *BR1_A_Adr*, *BR2_A_Adr* and *BR2_B_Adr* are the address signals on port A of *BR1,* port A of *BR2*, and port B of *BR2* respectively. A Block RAM is enabled or disabled by assigning '1' or '0' to its enable signal.

From Figure 7a, when the value of *base* is within the span of *p1*, the appropriate port on *BR1* is enabled and accessed while the relevant port on *BR2* is disabled. The reverse is the case when *base* is no longer within the span of *p1*, i.e. within the span of *p2*. This simple example could be extended to cases in which more than one segment makes up a GMO and each segment has more than 2 partitions. A formal description of this approach is shown in Figure 7b. Figure 7c depicts the base pointer implementation of the GMO shown in Figure 6.

5.2 Distributed Pointer Approach

In this approach, each partition is handled separately, starting with the first partition in a segment. Local pointers equal in length to the length of each partition are created. As long as the enable signal of Block RAM for a partition is high, memory access is started at its first position using its pointer and continues incrementally when there are valid data until its full length is reached. During this period, the partition ensures its enable signal is re-asserted while the enable signals of the neighbouring partitions of the same segment are kept low. Controls are transferred to the next partition of similar segment when the upper limit of the partition is reached. If however, the partition is the last in the segment, controls are transferred to the first partition. Since the address buses of partitions on Block RAMs provide appropriate bit vectors to cover their entire lengths, they are used as the local pointer. In this approach, the enable signals of all the first partitions are set to high at start-up to ensure that memory accesses start with the first partitions. Figure 8a depicts this approach. A simplified case of a GMO consisting of a single segment with two partitions *p1* and *p2* allocated on Block RAMs *BR1* and *BR2* respectively is considered in Figure 8a. The signals in Figure 8 have similar meanings as in Figure 7. Figures 8b and 8c show formal description and implementation of the GMO depicted in Figure 6 using this approach. Since the 640 by 16 partition is the only one in its segment, it is always enabled and the address is reset to 0 when it reaches its the upper limit.

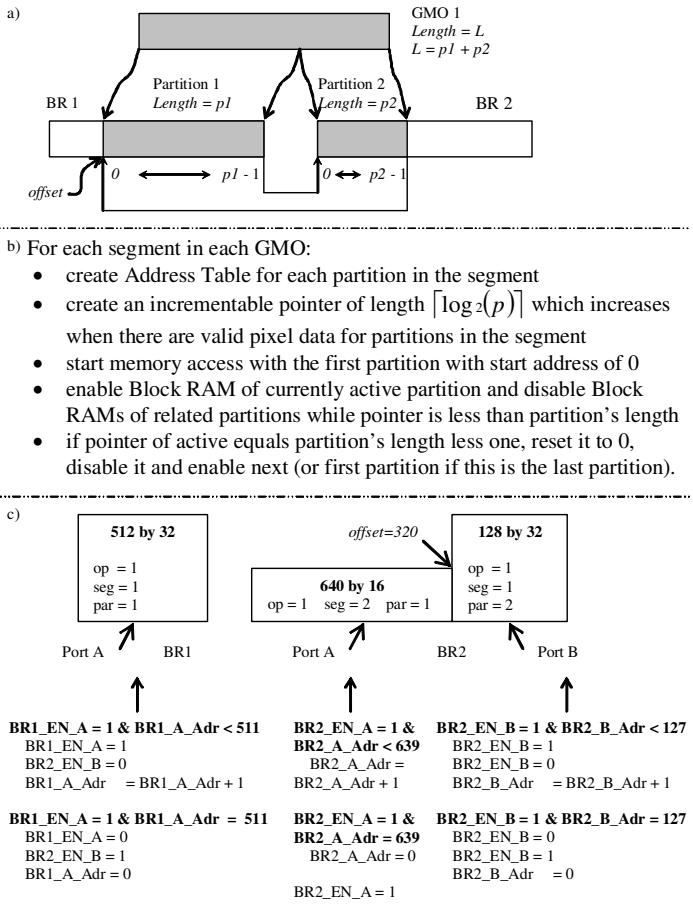


Figure 8. Distributed Approach.

6 Experimental Results

The work presented in this paper incorporates address generation using the two approaches described above. The allocation algorithm, address generation and VHDL code generation were implemented in C++ which takes FPGA technology files and memory requirements as input. The implementation was simulated using the memory requirements of real-time video processing design cases [17]. The first design case was a spatio-temporal median filter with a neighbourhood of seven frames and two line buffers. Two instances of this design case were considered. The first, (1-1), being a VGA frame with 24-bit RGB pixels and a 640 frame length, while the second, (1-2), was a PAL frame with an 8-bit gray scale pixel and a 708 frame length. The second design case was a machine vision system with a median filter, segmentation and three 1-bit morphological operations. Also, in this design case two instances were considered. The first, (2-1), being an 8-bit gray scale with VGA resolution as the input video stream while the second (2-2) had a 12-bit gray scale with 1.3 MPixel resolution as the input video stream. Table 1 shows the summary of the memory requirements for the design cases considered.

Table 1. Memory requirement for design cases.

Design Case	#	Rows	Width	Length	Size (kbit)
Case 1-1	7	2	24	640	210
Case 1-2	7	2	8	708	77.4
Case 2-1	1	4	8	640	20.0
	1	1	19	256	4.75
	3	16	1	640	30.0
Case 2-2	1	4	12	1300	60.94
	1	1	21	4096	84.0
	3	16	1	1300	60.94

Tables 2 shows the results obtained using the implementation of the algorithm for allocating the design cases considered on Xilinx Spartan 3 FPGA.

Table 2. Allocation result of the algorithm on Spartan 3.

Design Cases	Minima	Block RAM	% minima
Case 1-1	14	14	100
Case 1-2	5	6	120
Case 2-1	4	5	125
Case 2-2	13	13	100

In the table the theoretical minima Block RAM required for allocation were estimated from equation (11) [17].

$$minimal = \left\lceil \frac{Size}{\text{size}(BRAM)} \right\rceil \tag{11}$$

where *Size* is the number of bits required by the design case, given in column 6 in Table 1, and the size of BRAM is the numbers of bits in one Block RAM, 16kbit (without parity) for Xilinx Spartan 3 [3]. Table 2 shows that allocation requirements were equal to the minima values apart from two of the design cases. The minimum value is calculated for allocation on a Block RAM with an infinite number of ports. The minimum value however only indicates the effectiveness of the allocation but not its feasibility, since it is not possible to have Block RAMs with an infinite number of ports. The implementation did not consider parity. The parity feature on Xilinx Spartan 3 FPGA increases the available Block RAM size by providing an additional bit for every 8 bits [3]. When taken into consideration the parity bit makes it possible to have width configurations that are multiples of 9-bit on the Block RAM. In this manner, 18Kbits of Block RAM size can be achieved instead of 16Kbits. This parity feature was not considered since it is specific to only some of the Xilinx FPGA families and not all FPGAs have this feature. From Table 2, the non-minimum result of the algorithm in design cases 1-2 and 2-1 is because if a design case has many operators in relation to the total storage requirement and/or the size of each Block RAM, the number of ports on each Block RAM will limit the allocation.

Table 3 shows the resources required to access the allocated memory objects for the design cases in Table 1, the number of Block RAM required for the allocations and the hardware operation frequency for the two approaches. Xilinx Spartan 3 FPGA was the target platform for implementing both of the above approaches.

Table 3. Comparison of the two approaches

	Case 1-1		Case 1-2		Case 2-1		Case 2-2	
	BP	Dist	BP	Dist	BP	Dist	BP	Dist
No. of 4 input LUTs:	653	994	334	356	155	191	560	804
No. of BRAMs:	14	14	6	6	5	5	13	13
Max. Frequency (MHz):	116	186	106	183	140	214	91	173
Frequency Comparison (%):	100	160	100	173	100	153	100	190

6.1 Complexity analysis

In estimating the complexity of the algorithm, the number of available Block RAMs, *K*, and the number of memory objects, *I*, after partitioning with respect to their width, play the major roles. Since the algorithm makes one iteration through the sets of Block RAMs and two iterations through the set of memory objects

as shown in steps 3, 4 and 13 in Figure 4, the allocation algorithm *AA* is a function of *K* and *I* and its complexity could be expressed as

$$AA(K,I) = \Theta(K) \cdot \Theta(I^2) \tag{12}$$

The algorithm is thus, at worse, of the third order of the larger of *K* and *I*. Implementation costs depend on the representations of the properties of the Block RAMs, memory objects and allocation objects, and the arithmetic and logic operations defined for them.

7 Discussions

Depending on the number of partitions relating to a GMO, address look-up tables are required to set the enable signals and the values of the address signals to the appropriate Block RAMs on which the element of the GMO currently being pointed at is allocated, while also disabling related Block RAMs. In the Base Pointer Approach, these accesses to the Block RAMs are centrally controlled at the GMO level using a pointer. Hence only one set of address look-up tables is required for each GMO. By contrast, in the Distributed Approach, each partition has its separate address look-up table that is unrelated to those of related partitions. The use of a partition's address look-up table depends on the value of its enable signal. Hence the total number of address look-up tables for one GMO depends on the number of partitions making up the GMO. This is evident by comparing Figures 7c and 8c. The first row of Table 3 confirms this. Thus the Base Pointer Approach yields more efficient use of hardware resources than the Distributed Approach. The differences in resource requirements are however marginal amounting to less than 3% of the available resources, for example, Xilinx Spartan 3 XC3S400 series [3].

Delays associated with the distribution of a single base pointer caused by long delays in the FPGA are eliminated in the Distributed Pointer Approach since each partition has one local pointer. The use of separate address look-up table for each partition in the Distributed Pointer Approach increases the speed of memory accesses and consequently, increases operating frequency. This is because all signals required for memory accesses are calculated simultaneously at the clock edge. As the third row in Table 3 shows, the Distributed Approach yields more rapid access to data than the Base Pointer Approach.

8 Conclusions

In this paper a heuristics based algorithm for allocation of memory objects to on-chip Block RAMs and two approaches for generating addresses to the access memory objects has been presented. The allocation algorithm assumes that memory objects required by an operator in an RTVPS can be grouped together to form global memory objects at the operator level. It also partitions the memory objects to ensure that their widths conform to those specified by the FPGA. The algorithm takes advantage of the dual port capabilities of the Block RAMs to achieve optimal allocations and the possibility of allocating a global memory object to as many Block RAMs as required. The results show of the allocation algorithm requires not more than 100% of the theoretical minimal value for most of the cases. However, minimum allocation cannot be guaranteed for all the cases since such allocation requires an infinite number of ports on a Block RAM.

The accessing approaches make the implementation of accesses to memory data trivial irrespective of their allocation. Two approaches, Base Pointer Approach and Distributed Pointer Approach have been presented and compared with respect to used hardware resources and performance. The results indicate that the Base Pointer Approach requires fewer resources than does the Distributed Pointer Approach. This resource reduction is however marginal when compared to the total capacity of the device. The Distributed Approach is at least 50% faster than the Base Pointer Approach for all the design cases considered.

Hence, employing this allocation algorithm in FPGA Block RAM allocation and the Distributed Approach maximises the use of FPGA Block RAMS and improves the processing power of FPGA as a reconfigurable platform for RTVPS implementation. Additionally, the heuristics nature of the algorithm means its implementation can be optimised for performance, and the automatic generation of addresses and VHDL code will simplify the implementation of real-time video processing systems.

Creating global memory objects greatly reduces the number Block RAMs therefore reducing the number of memory read/write operations required per operation in the real-time video processing system and hence minimising power consumptions in the FPGA.

Reference

[1] Gonzalez, R., and R. Woods, *Digital Image Processing*, 2nd edition, Addison-Wesley Pub., 2002.

[2] G. Liersch, and C. Dick, "Reconfigurable Gate Array Architectures for Real Time Digital Signal Processing", *Conf. Record of the Twenty-Eighth Asilomar Conf. on Signals, Systems and Computers,* Nov 1994, pp 1383 - 1387.

[3] XILINX, *Using Block SelectRAM+ Memory in Spartan-II FPGAs*, XAPP173 (v1.1), http://www.xilinx.com, Dec 2000.

[4] XILINX, *Using Block RAM in Spartan-3 FPGAs*, XAPP463 (v1.1.2), http://www.xilinx.com, July 2003.

[5] M. Doggett, and M. Meissner, "A Memory Addressing And Access Design for Real Time Volue Rendering", *Proc of IEEE Int. Symp. on Circt. and Syst.* June 1999, pp 344 - 347.

[6] D. Grant, P.B. Denyer, and I. Finlay, "Synthesis of Address Generators", *Digest of Tech. Papers of IEEE Int. Conf on Computer-Aided Design*, Nov 1989, pp 116 - 119.

[7] J. Seo, T. Kim, and P.R. Panda, "Memory Allocation and Mapping in High-Level Synthesis - An Integrated Approach", *IEEE Trans. on VLSI Syst.* Oct. 2003, pp 928 - 938.

[8] F. Balasa, F. Catthoor, and H.M. Man, "Background memory area estimation for multidimensional signal processing systems," *IEEE Trans. on VLSI Syst*. Vol 3, pp. 157 - 172, June 1995.

[9] L. Nachtergaele, F. Catthoor, F. Balasa, F. Franssen, E. D. Greet, H. Samsom and H. D. Man, "Optimization of memory organization and hierarchy for decrease size and power in video and image process systems", *IEEE Int. Workshop on Mem Tech, Design and Testing*, Aug 1995, pp. 82 - 87.

[10] M. Weinhardt, and W. Luk, "Memory access optimization for reconfigurable systems." *IEE Proc-Comput. Digi. Tech.*, vol. 148, no.3 May 2001, pp 105 - 112.

[11] L. Ramachandran, D. D. Gajski, and V. Chaiyakul, "An Algorithm for Array Variable Clustering", *In Proc ,Europ. Des. Test. Conf., Feb.1994,* pp. 262 - 266.

[12] M. Gokhale and J. Stone "Automatic Allocation of Arrays to Memories in FPGA Processors With Multiple Memory Banks, in *Proc. of the IEEE Symp. on Field-Programmable Custom Machines*, 1999, pp. 63-69.

[13] S. Jaewon, K. Taewhan, and P. R. Panda, "Memory allocation and mapping in high-level synthesis – an integrated approach", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems,* vol.11, no.5, 2003

[14] C. H. Gebotys, "Low energy memory and register allocation using network flow", in *Proc. of Design Automation Conf.*, 1997, pp. 435-40

[15] R. Leupers, and P. Marwedel, "Algorithms for Address Assignment in DSP Code Generation", *Digest of Tech. Papers of IEEE/ACM Int. Conf. on Computer-Aided Design*, Nov. 1996, pp 109 - 112.

[16] N. Sugino, H. Miyazaki, S. Iimuro, and A. Nishihara, "Improved Code Optimization Method Utilizing Memory Addressing Operation and its Application to DSP Compiler", *IEEE International Symposium on Circuits and Systems*, May 1996, pp 249 - 252.

[17] M. O'Nils, B. Thörnberg and H. Norell, "A Comparison between Local and Global Memory Allocation for FPGA Implementation of Real-Time Video Processing Systems", in *Proc of IEEE Int. Conf. on Signals and Electronics Systems,* Sept 2004.

[18] B. Thörnberg, H. Norell and M. O'Nils, "IMEM: An Object-Oriented Memory- and Interface Modelling Approach for Real-Time Video Processing Systems", *Proc. of the ECSI Forum on Design Languages,* 2002.

[19] H. Norell, B. Thörnberg and M. O'Nils, "Automatic Hardware Synthesis of Spatial Memory Models for Real-Time Image Processing Systems", *Proc. of the 21st Norchip Conf.,* Riga, Latvia, Nov 2003

FPGA based Surveillance and Control Computer for Customer Specific Applications

Niklas Lepistö, Mattias O'Nils
Department of Information Technology and Media
Mid Sweden University
Holmg. 10, 85170 Sundsvall, SWEDEN
niklas.lepisto@miun.se, mattias.onils@miun.se

Abstract

The decreasing price/performance ratio of FPGA-technology and the availability of devices with embedded processors, communication- and high-speed I/O-blocks makes reconfigurable System on Chip (rSoC) based designs more attractive for each FPGA-generation that reaches the market. Surveillance and control computers for heavy vehicles require highly adaptable connection capabilities to fit the connectivity requirements for a certain vehicle, including several video and communication channels. This require several customer specific versions of similar products that still needs to fulfill tight cost constraints, which is similar to those found in ordinary automotive industry. This paper describes work-in-progress to investigate if a rSoC architecture can offer the price/performance required for the selected applications area. This paper identifies the different research aspects involved in the project together with possible problems and opportunities.

1 Introduction

The costs involved in traditional ASIC based SoC designs often limits their use to applications for the mass market where the cost per device is reduced by the large quantities produced [1]. State-of-art FPGA devices provide a way to increase the integration level of designs without the huge NRE-costs and slow design process associated with ASIC SoC devices [2].

The increasing performance and reducing costs of FPGA devices has also made FPGA based solutions attractive for applications where ASICs normally would be used. These applications are usually found in products that are manufactured in such quantities where the unit cost would be similar for both ASIC and FPGA based designs. The choice between the two technologies would have to be based on the the differences in time to market and resulting product performance. The fast design turnaround time of FPGA based designs reduce time to market significantly compared to an ASIC [1]. An FPGA based design would however result in a higher power consumption and reduced speed making it unsuitable for some applications. FPGAs do not provide any possibility to integrate analog components making it necessary to provide such functionality by additional external components thus complicating the design, this could also entirely rule out an FPGA based solution in some cases. The production of an ASIC does usually include a certain risk due to the high cost of design iterations. Even if the unit cost for ASIC and FPGA solutions for a product are expected to be equal, an unexpected design revision could increase the unit cost for an ASIC based product significantly. Some FPGA vendors provide a relatively safe path between an FPGA based design an a structured ASIC [1]. A structured ASIC provides some advantages such as lower power consumption and higher speed compared to FPGAs,. The most important disadvantage would be the lack of reconfigurability.

FPGA based designs are an interesting alternative for products that may need to be customized for a specific application. The reconfigurable nature of an FPGA allows changes to be made in hardware after the product has been manufactured. This flexibility could be used to provide a single hardware platform that can be used for a large number of product configurations which would reduce the production costs. An FPGA can be used in different ways to increase the flexibility of a system. In some cases the entire system can be implemented in an FPGA resulting in a rSoC design while others would use an FPGA as a reconfigurable co-processor [3] or communication controller [4]. The research field of rSoC is very active and a lot of work remains to be done. This report describes work-in-progress to investigate if a rSoC architecture can offer the price/performance required for implementation of a computer system intended for heavy vehicles.

2 Target application

During the last years the use of computer systems in vehicles has expanded rapidly. Usually these systems are used for navigation and to present statistics or data for the different components in the vehicle. Automotive computers in

Figure 1. Timberjack harvester

tended for heavy vehicles differ slightly from the devices used in ordinary vehicles. Heavy vehicles such as forest harvesters and excavators are usually equipped with complex tool systems that are controlled by a computer. These tools are usually exchangeable which provides flexibility for the machines but also increases the requirements on the computers intended to control them. A computer for this purpose would require a large number of different connection possibilities to cover the wide variety of interfaces and bus standards. This would include interfaces normally used in industrial applications together with the usual interfaces associated with personal computers as well as video inputs used for surveillance cameras. The harsh environments where the computers usually would operate in requires a sealed enclosure which complicates cooling of the components. This increases the importance of a low power consumption. Heavy vehicles like harvesters are usually produced in small quantities compared to regular vehicles making it expensive to design a specific solution for each type of vehicle. The cost constraints are similar to those found in ordinary automotive industry where increased product costs can not be motivated by additional functionality. The increased configurability provided by a rSoC based system could make it an cost effective alternative for on-board computers intended for use in heavy vehicles.

3 Research challenges

3.1 Product configurability

By using a rSoC based design some of the components that differ between the different versions of a product can be implemented in the FPGA making them entirely reconfigurable. This does however require a supply of IP-cores for the components intended to be integrated. The manufacturers of FPGA devices usually offer a wide variety of comercial IP-cores. There also exists different open source alternatives of varying quality which usually are released under GNU GPL or similar license. This work includes investigation in which parts of a typical embedded system would be suitable for integration in a rSoC. The suitability would be based on available IP-cores , FPGA performance and total cost. The investigation of IP-cores would mainly focus on communication controllers that can be used for external communication but could also include a comparison of different soft-core processors.

The inceased flexibility provided by a SoF based design is often limited by hardware. Even though the components implemented in the FPGA are reconfigurable the electrical and physical interface used for external communication may not provide the same possibilities. Even though FPGAs usually support many different I/O standards there is no support for communication standards that are normally used for external communication[2]. A communication controller implemented in the FPGA would require an electrical interface for signal level conversion. The most practical solution would be to use an external device which provides a configurable electrical interface that can be used for a number of different communication standards. This kind of devices may be simple to design for low data rates but with high speed communication interfaces such as USB 2.0, FireWire or Fast Ethernet the differences in impedance and other electrical characteristics would be difficult to manage. The physical differences between connectors of different communication standards would also be a problem for a configurable communication device. One method to solve both these problems would be to use a modular hardware design. A module containing a connector and electrical interface for a specific standard could be attached to an expansion port, which is connected to the FPGA. This would increase the hardware design costs but would still be profitable compared to totally customized products. One of the tasks involved in this project is to analyze the appropriate level of FPGA integration for different communication standards.

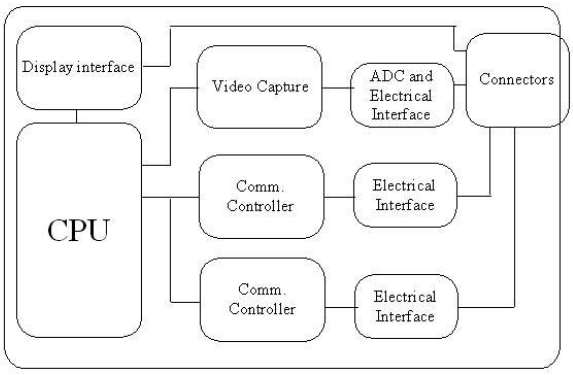


Figure 2. FPGA based system with configurable communication devices

An FPGA based system could be designed to be dynamically reconfigurable allowing a reconfiguration of hardware during operation[5]. This could be practical for systems that use hardware implemented audio/video filters or compression algorithms. The hardware requirements for dynamic reconfiguration depends on the system design. To make a dynamically reconfigurable rSoC where all components have been implemented in the FPGA, the FPGA would have to be partially reconfigurable[6]. This would not be required if the system has a separate CPU connected to an FPGA.

3.2 Component interconnection

Some research results show that the connection methods used between the components in ASIC SoC devices are far from optimal for implementation in FPGA based designs [7]. Studies has shown that wide data paths in FPGAs are very power consuming. Up to 60% of the power consumption in an FPGA is usually caused by wiring [8]. Industrial applications are often mounted in sealed enclosures for protective purposes. This makes it difficult to provide sufficient cooling increasing the importance of a low power consumption. Some reports indicate that serial serial connection interfaces can be an alternative for low speed on-chip communication and would reduce the power consumption significantly[7]. Some studies have also addressed the large wire delays in FPGAs, providing different methods to reduce and analyze these delays [9][10]. There still remains a lot of research in finding interconnection methods that are suitable for FPGA based designs.

4 Conclusions

Usually FPGA designs can be found in application where ordinary ASIC designs are not a economical option. FPGA technology can also be used to reduce production cost by improving the flexibility of a product. This paper has presented a planned investigation which is to determine if an rSoC based design would be suitable for an on-board computer intended for use in heavy vehicles. The investigation also includes a number of research subjects in the field of rSoC design that requires more work. Some of the intended research subjects are:

- Investigation of available IP-cores for different communication controllers and soft-core CPUs.
- Analysis of the appropriate level of FPGA integration for different communication standards.
- Research in suitable component interconnection methods for FPGA designs.
- Design test applications for dynamic reconfiguration.

References

[1] Zahiri, B., “Structured ASICs: opportunities and challenges”, 21st International Conference on Computer Design, 2003. Proceedings.

[2] Xilinx Inc., www.xilinx.com

[3] Luthra, M. Sumit Gupta Nikil Dutt Rajesh Gupta Nicolau, A. “ Interface synthesis using memory mapping for an FPGA platform” , Proceedings. 21st International Conference on Computer Design, 2003.

[4] Donchev, B. Hristov, M. “Implementation of CAN controller with FPGA structures”, CAD Systems in Microelectronics, 2003. CADSM 2003. Proceedings of the 7th International Conference.

[5] Zuim, R.L. Junior, C.J.N.C. Moreira, L.F.E.Fernandes, A.O. da Mata, J.M. da Silva, D.C., Jr. “Dynamic Reconfiguration Behavior Using Generic FPGAs and FPIDs”, Integrated Circuits and Systems Design, SBCCI 2003

[6] Mesquita, D. Moraes, F. Palma, J. Moller, L. Calazans, N., “Remote and partial reconfiguration of FPGAs: tools and trends”, Parallel and Distributed Processing Symposium, 2003. Proceedings. International.

[7] Andy S. Lee, Neil W. Bergmann, “On-chip Communication Architectures for Reconfigurable System-on-Chip” , Proceedings of Field-Programmable Technology (FPT), 2003.IEEE

[8] F. Li, D. Chen, L. He, J. Cong “Architectures evaluation for power-efficient FPGAs”, Proceedings of the ACM International Symposium on FPGA, Feb. 2003.

[9] Brown, S. Khellah, M. Vranesic, N., “Minimizing FPGA interconnect delays”, Design & Test of Computers, IEEE 1996

[10] Kannan P., Bhatia D., “Interconnect estimation for FPGAs under timing driven domains”, 21st International Conference on Computer Design, 2003. Proceedings.

Design of Electrical Architectures for Safety Cases

Industrial PhD project

Fredrik Törner

Table of contents

Who I am

- Fredrik Törner
- Master of Science in Electrical Engineering, Chalmers 2004
  - Specialized in the Dependable Computer Systems International Master Program at Chalmers
- Master thesis involved Fault Tolerance in a TTCAN based demonstrator at Volvo Cars
- Worked with Model Based Functional Development at Volvo Cars during the past year.
- Industial PhD student since 7/2 2005 at Volvo Cars

PhD project

- A joint project between Chalmers and Volvo Cars
- Supervisors will be:
  - Peter Öhman, Chalmers
  - Per Johannessen, Volvo Cars
- Finance: Volvo Cars, Swedish Automotive Research Program
- Timeplan: 80% research, 20% Volvo Cars other projects during 2005-2009

Background to the PhD project

Challenges, development of automotive electronics

"For 28 percent of our Mercedes owners (and 17 percent of BMW drivers), trouble started on day one…"

"In most cases the culprits could be found somewhere in the electronic network."

Autocar / Auto motor und sport


Made in Germany
Wolfgang König
28 per cent of Mercedes SL owners had trouble from day one

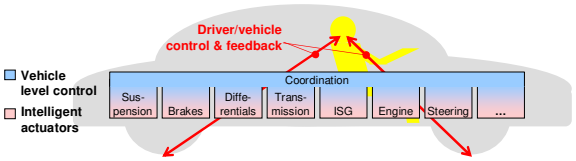
Future development of automotive systems

- Adaptive Cruise Control, Forward Collision Warning
- Electric Parking Brake, Electric Steering Column Lock

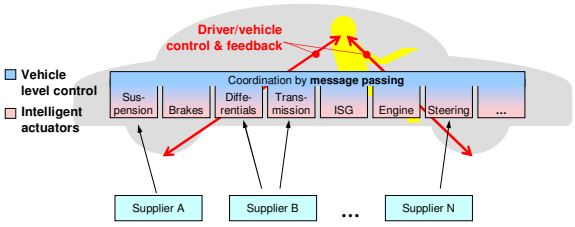
- Lane Keeping Aid
- Electric Power Assisted Steering, Electro-Hydraulic Brakes

- Collision Avoidance by Braking and by Steering
- Brake-by-Wire, Steer-by-Wire

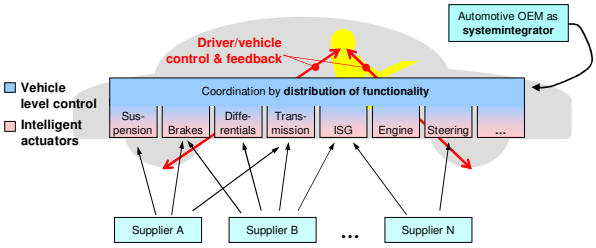
System solution



System solution – implementation today



System solution – implementation future



Summarized challenges

- Increased complexity in automotive functionality

- Distributed functionality

- Increased interaction between subsystem demands that the OEM will take the role as system integrator.

Legal requirements regarding ”Complex Electronic Vehicle Control Systems”

- *A description of the measures designed into the system, for example within the electronic units, so as to address system integrity and thereby ensure safe operation even in the event of an electrical failure*

- *Safety Case*

- Is valid for brake systems today (ECE R13H)
- Will soon be valid for steering systems (ECE R79H)

Safety

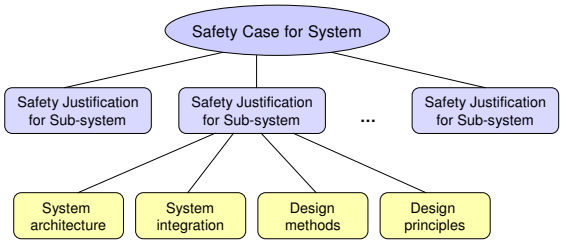
- Safety is a system property and has to be design as such
- Safety can be seperated in functional safety and system safety
- System safety implies that the system should not cause a serious situation

- *Safety Cases* are related to system safety

Safety Case

- A *Safety Case* argues that a system is safe and can be used as a statment for product liablility
- Existing legal requirements for *Complex Electronic Vehicle Control Systems* demands *Safety Cases*
- A *Safety Case* can be based on:
  - Fault analysis such as FMEA, FTA
  - Reference architectures
  - Design principles
  - Development processes
  - …

Safety Case



PhD project description

Research questions

How is a safety case best developed for distributed control systems in the automotive industry?

How can a safety case be developed to support the development of electrical architectures?

Relevance for Volvo

- Swedish automotive industry has high expectations from customers and owners.
- A *Safety Case* is required to be able to sell cars with electrical controled chassi systems.
- A *Safety Case* raises the opportunity to increase the system safety in itself through increased awareness.
- Future customer functions requires increased focus on safety.

Project deliverables

- Development process for *Safety Cases*
  - Integration of *Safety Justifications* to form a good *Safety Case*
- Adapted system architecture
- Method for system integration
  - Definition of fault containing areas
  - Fault handling at interfaces

Project deliverables cont'

- Designmethods
  - Risk analysis
  - Fault analysis methods
- Design principles
  - Reliable communication
  - Redundancy
  - Fault detection
- Case studies for verification and validation of results

Summary and questions

- *Safety Cases* will be increasingly important for the automotive industry due to legal requirements and more complex systems
- The project goal is to develop a validated process with development methods and design principles for Safety Cases within the automotive area.
- Questions?

Design of Electrical Architectures for Safety Cases

Industrial PhD project

Fredrik Törner
ftorner@volvocars.com

ADVANCED COMPONENT-BASED SOFTWARE ENGINEERING COURSE

GUEST LECTURES 2005- 02-23, MDH, ZETA Room

Lecture 1: 13:15-15:00 Wolfgang Weck: How is Eclipse Coming Along as a Component Framework?

LEcture 2: 15:15-17:00, Mikael Åkerholm: Real-Time Component-based Echnology: SaveCCM

Wolfgang Weck: How is Eclipse Coming Along as a Component Framework?

Eclipse is widely used as an (extensible) IDE. Starting with version 3, it is also advertised as a Rich Client Platform (RCP). To what extend is it also a component framework? A component framework not only needs to support extension with new components and interaction between them, it should also safeguard developers from certain mistakes that are easily made and hardly detected, if components are constructed independently and composed later by third parties. We look at the technology Eclipse provides but also into the "feature interaction" of component principles and open source.

Dr. Wolfgang Weck is an independent software architect based in Zurich, Switzerland. Besides his consulting work he is lecturing at various Universities of Applied Sciences. He has been leading numerous consulting and implementation projects and was the product manager of the BlackBox Component Builder while working for Oberon microsystems. He has conducted research in programming language support for component software and component specification at Åbo Akademi, Turku, Finland, and ETH Zurich, Switzerland. He has presented accepted and invited talks at various conferences, such as OOPLSA and FMCO and he is one of the co-organizers of the yearly WCOP workshops.

Mikael Åkerholm: Real-Time Component-based Technology: SaveCCM

This presentation describes a prototype component technology, intended for embedded control applications in vehicular systems. The technology has limited flexibility to facilitate future application of analysis of imporatant quality attributes in the domain, however, the
expressive power has been focused to the needs of vehicular systems. The presentation will conclude with suggestions for student projects related to the technology.

Mikael Åkerholm is a PhD student at MdH/IDE. He received a master's degree in computer science and engineering from Mälardalen University in 2003, and continued with PhD studies at the same department directly. Mikael has obtained his licentiate degree "A Software Component Technology for Vehicle Control Systems - Trade-Off Between Enginnering Flexibility and Predictability" in 2005. Mikael's research interests are component based software engineering, real-time, safety-critical, and embedded systems. He is participating in the SAVE project, which is a research project that tries to enable component based software engineering for safety critical vehicular systems.
