Robust Partitioning for Reliable Real-Time Systems

Reinhard Seyer¹, Christian Siemers², Rainer Falsett¹, Klaus Ecker², Harald Richter²
¹DaimlerChrysler Research, Post Box 71 01 62, D-60491 Frankfurt/Main, Germany,
(Rainer.r.Falsett|Reinhard.r.Seyer)@daimlerchrysler.com
²Technical University Clausthal, Institute for Computer Science, Julius-Albert-Straße 4, D-38678
Clausthal-Zellerfeld, Germany, (siemers|ecker|richter)@informatik.tu-clausthal.de

Abstract
Mechatronic systems request for high reliability, especially in the context of time where mostly hard real-time capabilities are mandatory. Maybe even stronger requirements regard the robustness against software failures and interdependences from erroneous tasks to others. This paper proposes the concept of robust partitioning for reliable real-time embedded systems. The concept consists of two parts, memory space protection and time protection. Memory protection is realized by already existing hardware and software mechanisms. For realizing temporal protection, a two-step timer interrupt system realizing an imprecise computation concept is proposed: If the execution of a module exceeds a certain time limit before the deadline, the first timer interrupt is triggered and a backup routine is started to produce an imprecise result in the remaining time until the second timer expires. This time protection concept shows significant advantages as compared to classical approaches for single, parallel and distributed systems. The paper gives an extended introduction into the concept and discussed first attempts for its realization.

Keywords: Mechatronic System, Reliability, Distributed Real-Time Systems, Time Supervision, Imprecise Computation

1 Introduction
In preceding papers [1, 2], an event-triggered approach was introduced and discussed to design efficient systems with real-time capabilities. The most important characteristics of this approach are the partitioning of interrupt service routines in a first and a main reaction part of the event-triggered system, which results in a deterministic behavior. The result from the first reaction part is called an imprecise reaction, and a second timer per interrupt channel is used to trigger the exceptional use of the imprecise value. A similar idea for single processor systems has been proposed in [3] and [4].
The event-triggered approach allowed improving schedulability in a significant way, and was used extensively to design systems for mechatronic and other applications. While the approach was very specific for event-driven systems, it showed its usefulness of hardware support for time supervision. The integrated timer provided the system with specific timing information combined with the triggering of specified activities.
The approach in the present paper treats the problem of supporting the system with timing information in a more general way: All tasks are defined with their logical and temporal behavior; on the hardware side additional components are defined to guarantee the temporal behavior. This approach can be compared to the supervising hardware inside real-time networks. In these systems, a special supervising hardware unit is responsible for avoiding all conflicts resulting from malfunction inside the node. The time supervision inside our approach works in a similar way and ensures the functionality concerning timing.
The paper is organized as follows: An introduction of robust partitioning including hardware enhancements is presented in detail in section 2. Section 3 describes the time supervision, section 4 the resulting development methodology and the enhancements to functional description. The summary and outlook to further research concludes this paper.

2 Key Elements of Robust Partitioning
Our primary goal is a simple and easy-to-understand design. Besides simplicity, there are other general principles that have to be observed. This comprises a transparent development process with clear specifications, a structured partitioning of functions, hard- and software modules, a well established and open communication process between all involved components and a reduction of all interdependencies.
These general requirements are in conflict with the increasing size of software and number of integrated units in embedded systems. The automotive area can be seen as a dedicated example for this tendency. Today a vehicle is under control of a distributed system comprising over 40 intelligent embedded electronic units for controlling nearly every car function, ranging from engine control, gear box, suspension, electrical power network, climate, seats, windows, security system, to communication to entertainment. It is obvious that these requirements result in a most complex system with many interactive units. Each embedded control unit (ECU) of such a system is set up by hardware and software, where the software volumes are in the range of hundreds of kilobytes up to even some megabytes.

A lot of problems are related to the fact that these units can only be expected to function reliably, if they have been very carefully tested against all conceivable scenarios. In practice they are assumed to work reliably and correct, if specified tests have been passed and no further error has been detected. Practical experience however shows that this is often not sufficient because of hidden errors slipping through all tests.

2.1 System Model

A distributed real-time system is composed of a variety of software components, as well as a variety of physical (hardware) components that govern the real-time performance. The physical components of a real-time system can be described by a set of computational resources and additional resources like interconnection networks, and other devices. Formally, the computational resources are a set of host computers. It is generally assumed that the properties of the computational resources are known. Regarding the software components, there is a logical view that distinguishes several levels of abstraction:

- The highest abstraction level represents the whole application software system. This is often referred to as “system” which, on the next lower level, is considered as a collection of units.
- A unit represents some part that can be logically separated from the total system. It is responsible for managing a part of the real-time system, such as a hardware component.
- Each unit consists of a set of tasks to be executed with known period or maximum frequency, and possible precedence constraints between them. The precedences for each path form a partial ordering on the set of tasks belonging to the path.

2.2 Intention of Robust Partitioning

The concept of “Robust Partitioning” (RP) addresses this test lack and was developed as an approach to regain simplicity and to provide functions to establish more reliable systems with much less effort. In order to do this, the RP concept comprises two main functions: The first is dedicated to a comprehensive memory protection. The purpose of the second function is to monitor precisely the real-time behavior of the system and to guarantee alignment to its specification. The concept intends to encapsulate the software due to memory space and execution time.

The first goal of RP is to define tight working limitations for the system units that can not be tunnelled or overcome by any means. There should even not be any chance or probability to violate the established limitations. This allows guaranteeing protection under all circumstances and for all possible situations, even impacts of an erroneous behavior of a unit to other components. In addition not only unintended violations should be prevented, but also those which might be intended but are not authorized.

The second goal aims to avoid negative impact of modern computer architecture elements like caches and protection systems to the predictability of execution times. The authors in [5] demand for new architectural concepts for processor-based hard real-time systems to enable realistic worst-case execution time ( WCET) assumptions. RP addresses this demand by introducing a supervision concept without negative impact on execution times. In the following sub-section we discuss the memory protection and time supervision inside RP that takes these requirements into account.

2.3 Memory Protection

State of the art of memory protection is already realized to some extent in memory management units (MMU) and in memory protection units (MPU). There are however some important differences between existing concepts and the approach of Robust Partitioning.

(a) Usually the MMU/MPU cooperates with a coarse-grain segmentation of the main memory. Safe systems require a fine-grain structuring which is realized in RP.

(b) The generally used block structure is too large to be effective for low level embedded systems, because 4k and even bigger blocks of 8 and 16k are not very appropriate to protect small memories.

(c) Within existing MMUs/MPUs, the protection functions are performed by tables that can be accessed at runtime. This means that the operating system has to maintain the tables and is able to change it. As a result there is a certain probability that the protection could be invalidated in cases of an error.

In the protection unit the RP-concept provides a separate protected memory for a table that contains the protection boundaries of the defined segments. It is generated at compile time and stored in a particular non-volatile memory, immune against EMI, bit errors, power supply interruptions and whatever else might happen. Favorably the table is allocated to a very reliable EEPROM or Flash-EPROM. The table is completely hidden from the execution and the operating system software and can not be altered at runtime. The table is fixed and can only be changed offline.
There is another disadvantage with most existing memory protection systems, and that refers to the fact that the protection tables have only small numbers of entries. If an address appears on the bus, which is not present in the table, it has to be reloaded. This takes time and delays the flow of the computation process. Obviously it is very complicated and in some cases even impossible to calculate or estimate this delay in advance. Evidently this is not compatible with the particular requirements of hard real-time systems.

Within the RP-concept the table is extended to cover all partitioned segments in one page. Since the table is stable, fixed and remains unchanged, there is no delay caused by any reloads or restructurings.

As a disadvantage, the RP is actually limited to compile-time defined systems.

2.4 Realization of the Memory protection

The memory of an embedded system is usually split into an instruction memory, which is in general allocated to a ROM-area, and into a data and stack memory mapped to a RAM (viz. Figure 1). The most import part of the memory to be protected is the RAM-area, and within the RAM-space the protection of the write accesses has highest priority.

![Figure 1. Memory Encapsulation](image)

As compared to data protection, stack protection appears to be more sophisticated and more complex, but there is a remarkable solution.

It is obvious that protection of all memory accesses (and not only the write cycles) would increase system security, because each access not intended and specified by the system designer, reveals itself by a violation of the protection boundary. An additional advantage is that these violations will be detected automatically without any further effort. This leads to a reduction of debugging and testing times, and hence allows much faster prototyping and generating a more reliable result.

Fragmentation of the data area is defined by a table as depicted in Figure 2 with a fixed number of entries. The designer might split the data area into a scratch pad, a module memory, variables data exchange areas and others. For a first attempt 128 entries might be an appropriate table size. To which extend this partition is useful should be found out by analyzing the trade-off between the advantage of the further splitting and the incurred cost.

![Figure 2. Protection Table](image)

Modern RISC- and superscalar architectures are accelerated by pipeline structures and caches in mutual dependence. The question how the use of caches allows a reliable system design [5] will be discussed in a later paper. The question discussed here is where to locate the protection unit. As the protection unit must control access to all addresses, data as well as instructions, it is mandatory to integrate this unit as close as possible to the CPU kernel, as depicted in figure 3. More or less, the protection will be part of the execution pipeline.

![Figure 3. Schematic view of the Integrated Protection Unit](image)

From the need to place the protection unit very close to the CPU follows that it can not be placed outside a processor chip. It has to be matched very closely to a given design and has to be fully integrated. This certainly leads to a problem because it can not be simply added to off the shelf processors. Therefore to test and to simulate the protection...
unit would be very complicated, and the results would not be very satisfying.

3 Time Supervision

Time supervision (or time protection) is the second key element of RP; its purpose is to monitor the execution times of program modules. It is state of the art that this task is performed by the operating system. Watchdogs, another well established time control concept, are realized in hardware, and offer a more sophisticated time control with greater functionality.

3.1 Time Supervision inside RP

The RP-concept takes the advantages of watchdogs into account and extends and upgrades them. Similar to watchdogs the supervision of timing is performed in an independent hardware entity.

While there is usually only one watchdog in even complex systems, which may of course be expanded to two or three, the number of the possible time protection sections in the RP-concept is determined by the number of memory protection entry points in the supervision table. On the other hand, the function to be performed by the watchdog is very simple: the execution time of a module is measured and continuously compared either against a given maximum value or against the task-specific deadline. Both techniques work in practical; the specific choice of comparison depends on the kind of real-time analysis and leads to minor differences.

As long as the module finishes before the run out time specified by the protection timer, nothing will happen. If the execution of a module takes longer than specified, it will be stopped and the control will be returned to the operating system.

Like the memory protection, time control system cannot be corrupted or skipped. This guarantees that all sections of the software are operating due the specification of the designed schedule.

To realize the time control system, another column is added to the protection table (figure 4). Its entries represent the execution times of all modules as defined by the system designer. The timers located in the protection unit measure the actual run time of the modules. These are continuously compared against the time limits, and in case of a violation, a non-masked interrupt will be initiated.

The observation of real-time requirements has to be controlled very rigidly. If one of the modules is expected to exceed its deadline it will be stopped and its execution cancelled. In order to give the stopped module a chance to produce a result, a second timer might be included. This leads to the concept introduced in [1], where a two-level interrupt-system was used to enable processing with imprecise but time-correct values.

<table>
<thead>
<tr>
<th>Entry Points ID</th>
<th>RAM</th>
<th>Stack</th>
<th>ROM</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>t_1</td>
</tr>
<tr>
<td>0x01</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>t_2</td>
</tr>
<tr>
<td>0x02</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>t_3</td>
</tr>
<tr>
<td>0x03</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>t_4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x7F</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>Upper address, Lower address</td>
<td>t_{128}</td>
</tr>
</tbody>
</table>

Figure 4. Protection Table with Time Supervision

This concept is adapted in our PR system. Two timers are used for each module, where the first timer would stop the execution of the related module some specified time before the deadline. It is proposed that in case of an abortion an emergency program is invoked, which is capable of producing simple (imprecise) result before the deadline is reached. This concept can be described as “hard determinism performed by weak logic”. The second timer controls the duration of the emergency program.

Both timers observe delays caused by hardware interrupt, i.e., in case of an interrupt, measuring the execution time is preempted, and it is resumed when the interrupt completed.

3.2 Consequences of RP for System Design

One general aspect of the Robust Partitioning approach is to enhance the reliability of the final system by using partitioning of system resources as a simplifying design concept. In addition, through the time supervision technique described above, remarkable additional effects provide the system designer with new possibilities.

Since system resources are not just limited but always turn out to be scarce, the design needs careful planning and realization. On the other hand, safe scheduling of hard real-time tasks requests for WCET estimates, and since the schedulability analysis for all parts of the system is based on these data, the resulting system will be largely overdimensioned [6]. In practical situations we may, however, assume that not all tasks system require for hard real-time behavior. The time supervision concept of RP takes advantage of such situation by allowing the system designer to perform a more detailed analysis. Tasks with strict time constraints and those with “soft real-time behavior” are analyzed independently. The hard real-time class considers computational times with respect to their WCET, while the computational times of the soft real-time class can be based on, e.g., the expected-case execution times (ECET). As reported in [7], ECET and WCET may differ by factors of more than 1000. By considering this the RP-based system enables a much more efficient system design.

One good example is the integration of caches inside modern microcontrollers and their impact on WCET...
analysis. As already mentioned, the speedup capabilities of a cache are either useless in hard real-time environments or contradict determinism. Depending on the cache replacement strategy, WCET estimations give in general very pessimistic results [5].

With the RP approach of controlling the execution time, it is possible to combine the advantage of a cache for speedup with the requirement of reliability for hard real-time tasks. During normal operation the acceleration of the cache can be used; if a time-out should occur due to a cache miss, the supervisor will detect this in due time, and react by initiating the fast emergency routine. From experience we know that this can be expected to happen only occasionally. This simple strategy keeps the program flow within the specified schedule, and therefore there will be no impact on the other tasks.

4 Development Methodology

On the basis of the RP concept, a modified development methodology can be imagined. Figure 5 shows the RP-based development process. The approach follows essentially the design process in embedded systems presented in [8, 1].

![Figure 5. RP-based development process](image)

The impact of RP is directly visible in the task classification and the WCET/ECET analysis, but besides this there are more principal arguments in favor of RP.

Due to the time and memory encapsulation capabilities the concept of RP can influence the structure of future software programs. Despite the fact that the object oriented approach is widely discussed [9], the protection scheme will enforce a software designer to keep very rigidly to it. Each object can be referred to as an encapsulated module with memory and timing characteristics. It is assumed that memory encapsulation can be supported by development tools. Particularly the linker will then be able to generate the memory boundaries, and with an appropriate interface it will be able to create a file with all required information for the memory protection unit inside RP.

The time protection unit can furthermore be used to support the development engineer with a function that allows to easily controlling his timing assumptions and calculations. For this the time protection unit is extended by a profiler, and by allowing external access to the internals of the protection timer. The software designer would then be able to online measuring the real code execution times during operation. The previously specified timing conditions may then be easily adjusted to the real speed. With this concept, all assumptions and calculations can be easily checked, changed and, within the test space, proved to be correct. The time protection function thus opens a new window in handling of timing.

It has already been mentioned that in case of a violation of the time limits an emergency program should be invoked by a non-masked interrupt. This new approach demands for some additional effort, because any task has to be considered in two different directions:

First the design has to be realized in a way that meets perfectly to all given requirements. In general the result will be a complex and sophisticated solution.

In addition emergency programs have to be designed. They should be comparatively simple and much shorter due to reduction of the very essential set of requirements. The emergency routines will hence run much faster at the price of a significant reduction concerning comfort, adaptability and flexibility.

In summary the two advantages will have a very positive impact to the development process.

A project development being distributed among several teams, it will always lead to problems in the integration phase. Bugs coming up take a lot of time and effort to clarify which team or person has caused them and is responsible for corrections. Obviously the encapsulation inside RP simplifies to clarify such questions without any further delay and investigation.

The other advantage regards the speedup of the debugging phase. Many errors that can be traced and fixed only with a huge effort will reveal themselves. This helps to speed-up the debugging and leads to a reduction of hidden errors.

5 Summary and Outlook

Robust Partitioning was developed as a system concept to encapsulate independent tasks from each other. Even more, mutual depending tasks like threads of one process should as well be encapsulated as much as possible. For this purpose, RP contains mainly concepts for memory protection and for time supervision. While the memory protection is a further development of existing protection systems, the time supervision appears to be new and contains some interesting features for hard real-time system design.

RP has impact on the processor architecture as well as the development process. Especially the memory protection subsystem must be integrated as close as possible to the processor kernel and appears as part of the execution pipeline. The time supervision itself could be coupled to
the processors in a more loosely manner, but there exist good reasons to integrate even this part as close as possible to the processors. One reason is that other system architecture elements like cache behavior or strategy must be incorporated in the time supervision unit, the other is that the interface between task switch and time supervision must also be incorporated. This interface should be as close as possible to the processor and as lean as possible.

The price to be paid on the development side is not zero, but the gained advantages are worth the effort. RP comprises for
- less effort for the testing and debugging,
- automatic error revelation,
- detection of hidden errors,
- controlled liability,
- combination of cache and reliability,
- support of software maintenance by error fixing in the field,
- increase of stability by control of error progression,
- improvement on resource usage (ECET versus WCET).

The next step will be to build up a microcontroller architecture with an integrated RP-unit. All actual efforts were done using external memory protection and time supervision. The projects were successfully, but as the external unit is limited by default, the full functionality is only testable on an advanced microcontroller architecture.

Acknowledgement: We thank the unknown referees for their valuable suggestions.

References