Robust Partitioning for Reliable Real-Time Systems

Reinhard Seyer, Rainer Falsett
DaimlerChrysler Research, Germany

Christian Siemers, Klaus Ecker, Harald Richter
Technical University Clausthal, Germany

1. Introduction

Mechatronic systems request for high reliability, especially in the context of

- time where mostly hard real-time capabilities are required
- robustness against software failures and interdependences from erroneous tasks to others

Siemers et al. 2003: event-triggered approach

main characteristics: two-step reaction

- **an imprecise reaction**
- **exception reaction in case of deadline failure**

Bettati et al 2003: similar idea for single processor systems

Present approach combines two concepts:

- time supervision as in the earlier approach

- ***Robust Partitioning:*** partitioning system resources leads to a simplified design

  Objective: enhanced the reliability because of simplified design

2. Key Elements of Robust Partitioning

General design requirements:

- simple and easy-to-understand design
- transparent development process with clear specifications
- structured partitioning of functions, hard- and software modules
- well established and open communication process between all involved components
- reduction of all interdependencies

These requirements are in conflict with

- the increasing size of software
- number of integrated units in embedded systems

Example automotive area:

Today's vehicle is under control of a

complex distributed system with many interdependencies

for controlling nearly every car function

engine control, gear box, suspension, electrical power network, climate, seats, windows, security system, entertainment

Assumption in practice: control is reliable and correct, after specified tests have been passed

Experience in practice: hidden errors slip through tests

2.1 System Model

A distributed real-time system is composed of

- **physical (hardware) components**
- **software components**

that govern the real-time performance

Physical components:

set of computational resources with known properties,

additional resources like interconnection networks,

and other devices

Software components:

The logical view distinguishes *several levels of abstraction:*

- Highest abstraction level: the whole application software system, referred to as "system"

  On the next lower level, the system is considered as a *collection of paths*

- A path represents some part that can be logically separated from the system

- Each path consists of a set of tasks with

  - known ***period*** or ***maximum frequency***

  - and possible ***precedence constraints*** between them

2.2 Robust Partitioning

Robust Partitioning comprises two main functions:

- comprehensive ***memory protection***
- precise monitoring the ***real-time behavior***

Goals of Robust Partitioning :

- define tight working limitations for the system units that can not be tunneled or overcome
- guarantee protection under all circumstances and for all possible situations
- avoid negative impact of modern computer architecture elements to the predictability of execution times

3. Memory Protection

3.1 Differences

between existing memory protection concepts and the proposed ***Robust Partitioning*** :

(a) Usually the MMU/MPU cooperates with a coarse-grain segmentation of the main memory

Robust Partitioning realizes a fine-grain structuring as required in safe systems

(b) Most existing memory protection systems are based on several tables with the requirement of table swapping at run time

Robust Partitioning

- provides a single table to cover all partitioned segments in one page
- the table is stored in a separate protected memory module and completely hidden from the execution and the operating system software
- the table cannot be altered at runtime

(c) As a disadvantage, the ***Robust Partitioning*** is actually limited to compile-time defined systems

3.2 Realization of the Memory Protection

Memory Encapsulation: Usually the memory is split into instruction, data, and stack memory:

	RAM				RAM				ROM	
Data	0x0000	Thread 1		Stack	uSL_1	Thread 1		Instructions	0x0000	Thread 1
	…				…				…	
	0x0022				ISL_1				0x007F	
	0x0023	…			uSL_2	…			0x8023	…
	…				…				…	
	0x003A				ISL_{n-1}				0x8234	
	0x003B	Thread n			uSL_n	Thread n			0x8235	Thread n
	…				…				…	
	0x006C				ISL_n				0xEF3A	
				uSL = upper Stack limit						
				ISL = lower stack limit						

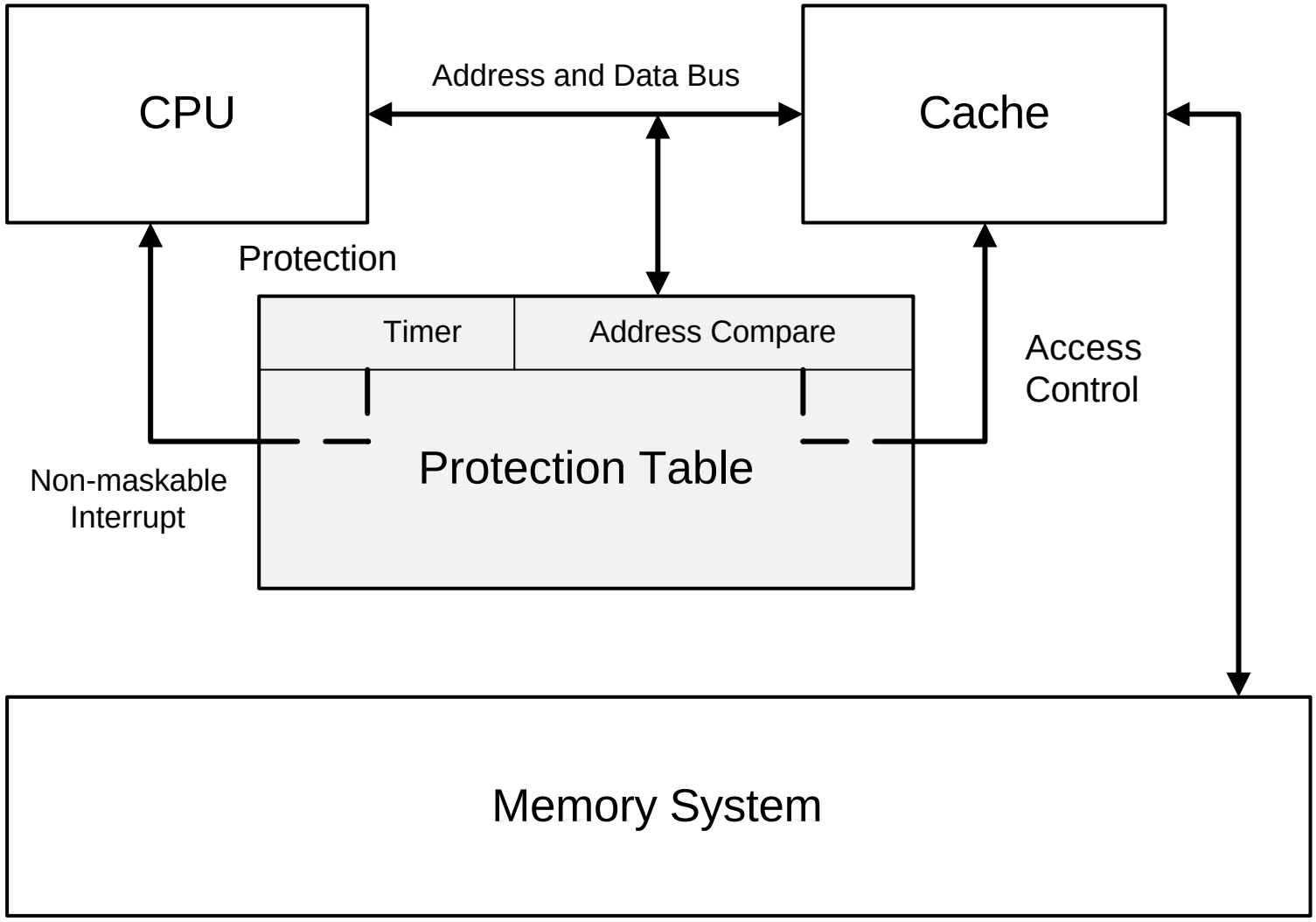
Memory fragmentation defined by a protection table:

Entry Points ID	RAM		ROM
	Data	Stack	
0x00	Upper $address_1$ Lower $address_1$	Upper $address_1$ Lower $address_1$	Upper $address_1$ Lower $address_1$
0x01	Upper $address_2$ Lower $address_2$	Upper $address_2$ Lower $address_2$	Upper $address_2$ Lower $address_2$
0x02	Upper $address_3$ Lower $address_3$	Upper $address_3$ Lower $address_3$	Upper $address_3$ Lower $address_3$
...	...	...	...
0x7F	Upper $address_{128}$ Lower $address_{128}$	Upper $address_{128}$ Lower $address_{128}$	Upper $address_{128}$ Lower $address_{128}$

The memory protection unit

- must control both, **write and read** accesses to data and instructions
  - **for increased system security**
  - **for automatic detection of access errors**
    - … leads to reduced debugging and testing times
    - … hence allows much faster prototyping
- should be closely located to the CPU kernel
  - **for reducing time delays**

Schematic view:



4. Time Supervision

Time supervision is the second key element of ***Robust Partitioning***

→ applies a **two-level interrupt-system** Siemers et al. (2003)

→ to enable processing with **imprecise but time-correct values**

Two timers (watchdogs) are used for each module:

- The **first timer** terminates the execution some specified time before the deadline

  an emergency program is invoked, for producing a simple (imprecise) result before the deadline expires

- The **second timer** controls the duration of the emergency program

To realize the time control system, a column with execution time limits is added to the protection table:

Entry Points ID	RAM		ROM	TIME
	Data	Stack		
0x00	Upper address$_1$ Lower address$_1$	Upper address$_1$ Lower address$_1$	Upper address$_1$ Lower address$_1$	t_{11} t_{12}
0x01	Upper address$_2$ Lower address$_2$	Upper address$_2$ Lower address$_2$	Upper address$_2$ Lower address$_2$	t_{21} t_{22}
0x02	Upper address$_3$ Lower address$_3$	Upper address$_3$ Lower address$_3$	Upper address$_3$ Lower address$_3$	t_{31} t_{12}
...	...	...	...	...
0x7F	Upper address$_{128}$ Lower address$_{128}$	Upper address$_{128}$ Lower address$_{128}$	Upper address$_{128}$ Lower address$_{128}$	$t_{1\ 128}$ $t_{1\ 128}$

5. Development Methodology

Design process for embedded systems:

Marwedel (2002), Siemers et al. (2003)

- Each object is regarded as an encapsulated module with memory and timing characteristics
- Particularly the linker will then be able to generate all required information for the memory protection unit inside ***Robust Partitioning***
- The time protection unit can furthermore be used to support the development engineer controlling the timing assumptions

6. Summary and Outlook

This presentation proposes the concept of ***Robust Partitioning*** for reliable real-time embedded systems

Objectives of ***Robust Partitioning***

- separate tasks from each other
- memory protection and time supervision

Robust Partitioning offers features

- less effort for testing and debugging
- automatic error revelation
- detection of hidden errors
- combination of cache and reliability
- support of software maintenance by error fixing in the field

Future work is building up a micro-controller architecture with integrated Robust Partitioning