

# Safety-critical Control in Mixed Criticality Embedded Systems

An evaluation of the EMC<sup>2</sup> development platform used in  
vehicle platooning

Master Thesis  
Royal Institute of Technology  
Stockholm, Sweden

Emil Hjelm  
`emilhje@kth.se`

April 7, 2017



Master Thesis MMK2017:Z MDAZZZ

Safety-critical Control in Mixed Criticality  
Embedded Systems

Emil Hjelm

Approved: (datum)	Examiner: Martin Törngren	Supervisor: Bengt Eriksson
	Uppdragsgivare: Alten	Kontaktperson: Detlef Scholle

## Abstract

Modern automotive systems contain a large number of Electronic Control Units, each controlling a specific system of a specific criticality level. To increase computational efficiency it is desired to combine multiple applications into fewer ECUs, this leads to mixed criticality embedded systems. The assurance of safety critical applications not being affected by non-critical applications on the same system is crucial.



Examensarbete MMK2017:Z MDAZZZ

Säkerhetskritisk kontroll i blandkritiska inbyggda  
system

Emil Hjelm

Godkänt: (datum)	Examinator: Martin Törngren	Handledare: Bengt Eriksson
	Uppdragsgivare: Alten	Kontaktperson: Detlef Scholle

## Sammanfattning

Denna del kommer att innehålla en sammanfattning av arbetet på svenska.

# Preface

Credit where credit is due.

Emil Hjelm  
Stockholm

# Contents

<b>Preface</b>	<b>iii</b>
<b>Abbreviations</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.1.1 Definition of safety-critical systems . . . . .	2
1.1.2 Different levels of criticality . . . . .	2
1.1.3 EMC <sup>2</sup> development board . . . . .	3
1.1.4 Platooning . . . . .	3
1.2 Problem statement . . . . .	3
1.3 Purpose . . . . .	4
1.4 Goals . . . . .	4
1.4.1 Team goal . . . . .	5
1.4.2 Individual goal . . . . .	5
1.5 Scope . . . . .	5
1.6 Research design . . . . .	5
1.7 Ethical considerations . . . . .	6
<b>2 State of the art</b>	<b>7</b>
2.1 Mixed criticality systems . . . . .	7
2.1.1 Economical benefits of MCS . . . . .	7
2.1.2 Sharing processor . . . . .	8
2.2 Standards . . . . .	10
2.2.1 IEC 61508 . . . . .	10
2.2.2 ISO 26262 . . . . .	11
2.2.3 AUTOSAR . . . . .	12
2.3 Mixed criticality platform solutions . . . . .	13
2.3.1 Hypervisors . . . . .	13
2.4 Platooning . . . . .	13
2.4.1 Benefits of platooning . . . . .	13

2.4.2	Safety requirements for platooning . . . . .	14
<b>3</b>	<b>Current system</b>	<b>15</b>
3.1	Soft overview . . . . .	15
3.2	Hardware . . . . .	15
3.3	TrustZone . . . . .	16
3.4	Operative systems . . . . .	17
3.5	Virtual Machine Monitor . . . . .	17
3.6	Build procedure . . . . .	19
<b>4</b>	<b>System design</b>	<b>22</b>
<b>5</b>	<b>Implementation</b>	<b>23</b>
<b>6</b>	<b>Results</b>	<b>24</b>
<b>7</b>	<b>Discussion</b>	<b>25</b>
7.1	Information retrieval . . . . .	25
7.2	Utilization . . . . .	25
<b>8</b>	<b>Future work</b>	<b>26</b>
8.1	MCS using virtualization . . . . .	26
8.2	MCS using other means of partitioning . . . . .	26
8.3	Amount of criticality levels . . . . .	26
8.4	Economical benefits for pursuing MCS . . . . .	27

# List of Figures

2.1	Percentage of schedulable tasks. [7]	10
2.2	AUTOSAR. [5]	13
3.1	Overview of the interfaces between the PS and PL regions. [28]	16
3.2	Flowchart of the boot sequence of the CPU. [28]	18
3.3	Overview of the MCS in place. [28]	19
3.4	System build procedure. [28]	21

# List of Tables

2.1	ASIL as a function of severity, probability and controllability. .	11
2.2	ASIL cost heuristics. . . . .	12



# Abbreviations

Abbreviation	Description
ECU	Electronic Control Unit
MCS	Mixed Criticality System
EMC <sup>2</sup>	Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments
RTOS	Real-Time Operating System
GPOS	General Purpose Operating System
FPGA	Field Programmable Gate Array
SIL	Safety Integrity Level
ASIL	Automotive Safety Integrity Level
DAL	Development Assurance Level
VMM	Virtual Machine Monitor
EMC <sup>2</sup> DP	EMC <sup>2</sup> Development Platform
RM	Rate Monotonic
DM	Deadline Monotonic
FP	Fixed Priority
EDF	Earliest Deadline First
AUTOSAR	AUTomotive Open System ARchitecture

# Chapter 1

## Introduction

This chapter will introduce the subject of mixed criticality embedded systems and the EU project "EMC<sup>2</sup>" to the reader.

### 1.1 Background

Today, modern automotive systems contain around 70-100 Electric Control Units (ECU)s [18]. Each ECU controls a subsystem of a specific criticality level such as safety-critical anti-lock brake system, or non-critical entertainment systems [23]. Having the ECUs isolated ensures that the numerous critical and non-critical applications do not interfere with each other, thus it is a simple task to certify an individual ECU. However, this approach leads to an inefficient use of system resources and expensive system implementation [10]. In order to lower the cost of the collective system and increase system efficiency (utilization), applications of different criticality levels can be integrated into a single multicore platform, leading to a Mixed Criticality System (MCS). However, this approach increases system complexity, and hinders the certification of safety-critical systems [28]. In order to facilitate the design, test, and certification of such systems, spatial and temporal partitioning can be used in the architecture of the system as described by [28].

Protecting the integrity of a component from the faults of another is desired in all systems hosting multiple applications. However, it is of higher significance if the different applications have different criticality levels. Without such protection all components on the same system would need to be engineered to the standards of the highest criticality level, potentially massively increasing development costs [10].

The EU project "Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments" (EMC<sup>2</sup>) was founded in order to "find solutions for dynamic adaptability in open systems, provide handling of mixed criticality applications under real-time conditions, scalability and utmost flexibility, full scale deployment and management of integrated tool chains, through the entire lifecycle" [23].

### 1.1.1 Definition of safety-critical systems

The term "safety-critical system" has many definitions, most quite similar. Most definitions relate to systems with the potential to harm humans if the system malfunctions. According to [13] it is defined as "A system in which any failure or design error has the potential to lead to loss of life." Further, [11] defines safety-critical systems as "A computer, electronic or electromechanical system whose failure may cause injury or death to human beings." A Wikipedia article, [25], defines a safety-critical system (or "life-critical system") as a system whose failure or malfunction may result in one (or more) of the following outcomes:

- death or serious injury to people
- loss or severe damage to equipment/property
- environmental harm

In this thesis, a safety-critical system will be defined as "a system whose failure may cause injury or death to human beings."

### 1.1.2 Different levels of criticality

Different names of levels of criticality are typically Safety Integrity Level (SIL), Automotive Safety Integrity Level (ASIL) and Development Assurance Level (DAL). The IEC 61508 standard [15] defines four different levels and the ISO 26262 standard [16] and the DO-178C standard [12] define five different levels each. These levels range from low or no hazard up to life-threatening or fatal in the event of a malfunction requiring the highest level of assurance that the dependent safety goals are sufficient and have been achieved.

The number of criticality levels in the implementation part of this thesis will be restricted to two: "safety-critical" and "non-critical". This is due to the constraints presented in section 1.1.3 and 1.5.

### 1.1.3 EMC<sup>2</sup> development board

As a part of the EMC<sup>2</sup> project, Alten has developed a system for handling applications of mixed criticality on the same piece of hardware. The MCS developed at Alten is implemented on a Xilinx Zynq-7000 [27]. The development board is called EMC<sup>2</sup> Development Platform, or EMC<sup>2</sup>DP. It employs two operating systems to handle applications of different criticality. A General Purpose Operating System (GPOS) for non-critical applications and a Real-Time Operating System (RTOS) for safety-critical applications. A Virtual Machine Monitor (VMM) is used to alternate between the two.

Resources on the board are separated between safety-critical and non-critical via ARM TrustZone [3].

For more detailed information, see section 3 or the report by Zaki [28].

### 1.1.4 Platooning

”The platooning concept can be defined as a collection of vehicles that travel together, actively coordinated in formation. Some expected advantages of platooning include increased fuel and traffic efficiency, safety and driver comfort” [9].

## 1.2 Problem statement

An ideal MCS ensures partitioning between different criticality levels while still sharing resources efficiently. This leads to the underlying research question:

- ”How, in a disciplined way, to reconcile the conflicting requirements of partitioning for safety assurance and sharing for efficient resource usage?” [10]

The MCS developed at Alten (EMC<sup>2</sup>DP) 1.1.3 switches Operative System (OS) to enable partitioning between safety-critical and non-critical applications, which takes about 2  $\mu s$ . This mode switch introduces additional deadlines which makes processor scheduling more difficult.

To evaluate the performance of the system, a distance keeping control algorithm for platooning will be implemented on it. A demonstrator will be constructed in the form of a RC car capable of following a vehicle in front

of it at a specified distance. If the lead car exceeds a predefined maximum speed or deviates from the road, the following car should not exceed the maximum speed. The performance of the embedded controller and the control algorithm will be measured during heavy non-critical computational load, and without any non-critical load altogether.

It should be verified that no matter the computational load and eventual crashes of the Linux based non-critical system, the distance keeping algorithm on the RTOS never crashes. It should also be investigated at how high frequencies the control algorithm can operate while still maintaining functionality on the GPOS.

This problem leads to the research question:

- How well can a safety-critical control system perform when implemented on a mixed criticality system using virtualization?

alternatively:

- Is virtualization an efficient approach when trying to reconcile the conflicting requirements of partitioning for safety assurance and sharing for efficient resource usage when implementing a safety-critical control system?

## 1.3 Purpose

Reducing the amount of computers in automotive systems would have many effects. Manufacturing costs would decrease and with fewer physical components maintenance costs would also decrease. However, the system complexity would increase and thereby increasing time and cost to design the system.

SafeCOP (Safe Cooperating Cyber-Physical Systems) is an European project that targets cyberphysical systems-of-systems whose safe cooperation relies on wireless communication [19]. SafeCOPs Use Case 3 (UC3) regarding "Vehicle control loss warning" together with the EMC2 goals tie well in with the problem statement and use case described in 1.2.

## 1.4 Goals

In this project there are both team goals and individual goals that do not always necessarily align with each other.

### **1.4.1 Team goal**

The team consists of five master thesis students. The students areas of work are: control theory and system modeling, data aggregation, safety-critical communication in MCS, lane detection and finally safety-critical control in MCS. Together the team will build a vehicle capable of following a vehicle ahead of it while keeping inside road markers.

### **1.4.2 Individual goal**

Verify quantitatively the performance of safety-critical distance keeping controller, see section 1.6. Solve the problems described in section 1.2.

## **1.5 Scope**

The work of this thesis and the implementation on the demonstrator will build upon the work of Youssef Zaki [28].

The embedded computer is constrained to the Xilinx Zynq-7000 SoC [27].

The architecture of the cooperative adaptive cruise control algorithm will be designed by Daniel Roshanghias.

## **1.6 Research design**

The plan is to conduct an confirmatory investigation using quantitative data/operations with a deductive approach. This is a quantitative research method where data is gathered during experiments and from simulations of the environment [14].

The position of the demonstrator will be read by a separate sensor of the same type as the one on the demonstrator. The performance of the control system and the embedded controller will be measured and compared with the same system without any non-critical computational load. This will also be done for a simulation of the system. The measures regarding control system performance will consist of

- Response time
- Overshoot

- Settling time

Data points for the performance of the embedded controller will be extracted from the RTOS, and the measures will consist of

- Missed deadlines
- CPU utilization

## 1.7 Ethical considerations

When designing a MCS it is crucial to ensure that errors made by a lower criticality application cannot propagate to higher criticality applications. This could have catastrophic consequences. Because of this the requirement of partitioning must have higher priority than the need of sharing.

# Chapter 2

## State of the art

This chapter will go through relevant articles and already known knowledge on the subject of mixed criticality systems, vehicle platooning and safety standards in the automotive industry.

### 2.1 Mixed criticality systems

A MCS is achieved by letting applications of different criticality share resources. These resources could be the processor, memory, peripherals, input/output ports etc. The most explored area is sharing the CPU between multiple criticality levels [10]. The benefit of combining previously distributed systems is higher resource efficiency, which leads to economical benefits.

#### 2.1.1 Economical benefits of MCS

Potential benefits with pursuing MCS as opposed to distributed systems are reduced physical space required, reduced weight, reduced heat generation, reduced power consumption and reduced production costs [10]. This would all ultimately lead to economical benefits.

Potential downsides are increased complexity which could lead to higher system design costs. Building applications on the same platform to share resources could require engineering teams to work more closely together, potentially leading to administrative difficulties and costs. This needs to be investigated and could vary from industry to industry. To combat the potential downsides, the EMC<sup>2</sup> project aims at creating platforms for easier



development of MCS.

The EMC<sup>2</sup> project lists several goals [24]:

- Reduce the cost of the system design by 15%
- Reduce the effort and time required for re-validation and re-certification of systems after making changes by 15%
- Manage a complexity increase of 25% with 10% effort reduction
- Achieve cross-sectorial reusability of Embedded Systems devices and architecture platforms that will be developed using the ARTEMIS JU results.

### **2.1.2 Sharing processor**

To deal with many different tasks needing processor time, different schedulers can be used to appropriately distribute processor time among the tasks.

#### **Conventional scheduling**

Fixed priority Deadline monotonic Rate monotonic Earliest deadline first Round robin

#### **Mixed criticality scheduling**

The area of sharing the processor in MCS was first explored by Steve Vestal [22] in 2007. His paper showed that neither Rate Monotonic (RM) nor Deadline Monotonic (DM) priority assignment was optimal for MCS; however Audsley's optimal priority assignment algorithm [4] was found to be applicable.

In 2008 Baruah and Vestal [8] showed that EDF (Earliest Deadline First) does not dominate FP when criticality levels are introduced, and that there are feasible systems that cannot be scheduled by EDF.

One MCS scheduling algorithm is Criticality Monotonic Priority Ordering (CrMPO). Tasks are assigned priorities first according to criticality (highest criticality first) and then according to deadline (shortest deadline first). Static Mixed Criticality with no run-time monitoring (SMC-NO) is the scheduler that was Vestal's original approach [22]. Another scheduler is SMC with run-time monitoring (abbreviated only as SMC). Yet another scheduling algorithm is Adaptive Mixed Criticality (AMC), described Baruah, Burns and

Davis [7]: "To summarise the main difference between SMC and AMC, in SMC any LO-critical task is descheduled if it executes for more than  $C(LO)$ . While in AMC, all LO-critical tasks are descheduled if any job (from any task) executes for more than  $C(LO)$ . If a HI-critical job executes for more than  $C(LO)$  (but no greater than  $C(HI)$ ) then, under SMC, LO-critical tasks continue to execute but may miss their deadlines; but under AMC they stop executing."

To evaluate the performance of the different scheduling algorithms Baruah, Burns and Davis [7] tested the scheduling algorithms AMC, SMC and CrMPO for scheduling sporadic tasks of a taskset of 20 tasks where on average 50% where of high criticality and 50% where of low criticality. The tasks of high criticality where allowed an execution time that was twice its low criticality execution time. The comparison of the performance of the schedulers can be seen in Figure 2.1. In the graph the UB-H&L line bounds the maximum possible number of schedulable task sets.

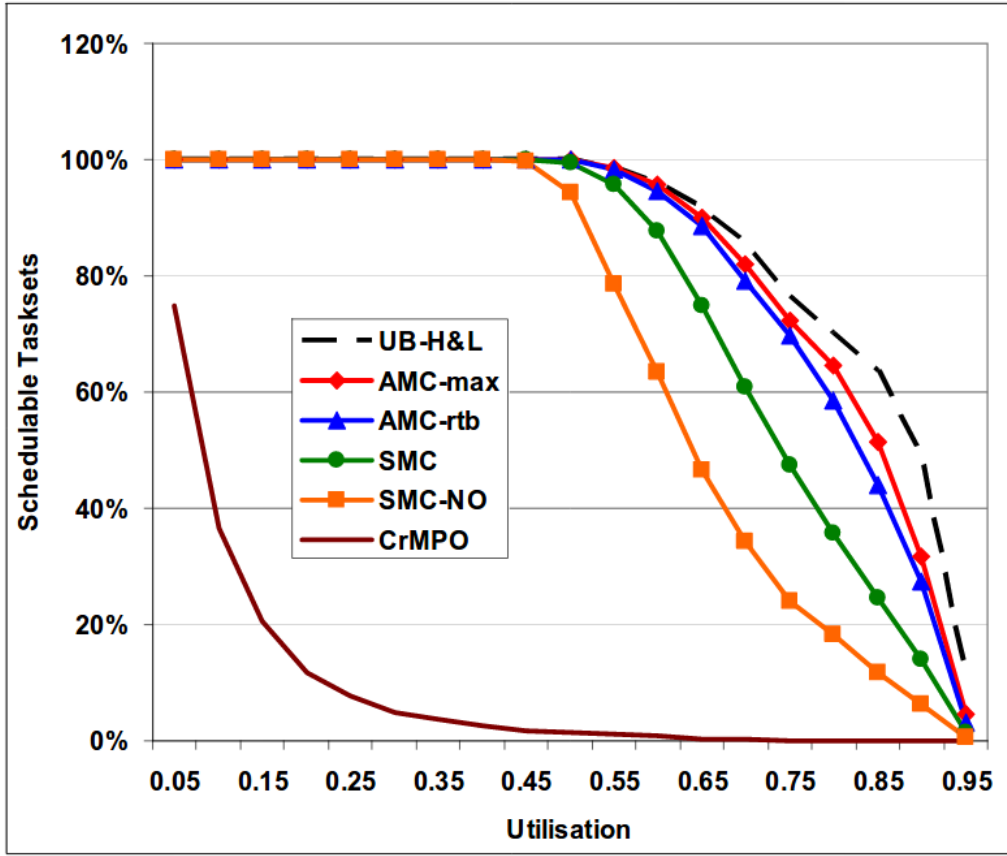


Figure 2.1: Percentage of schedulable tasks. [7]

For a more complete review of work done on MCSs with a shared processor, see the paper by Burns [10].

## 2.2 Standards

Safety practices are becoming more regulated as industries adopt a standardized set of practices for designing and testing products.

### 2.2.1 IEC 61508

IEC 61508 [15] is intended to be a basic functional safety standard for electrical and electronic systems applicable to all kinds of industry. It defines four different safety integrity levels, SIL 1 being the least dependable up to SIL 4 which is the most dependable level.

### 2.2.2 ISO 26262

ISO 26262 [16] addresses the needs for an automotive-specific international standard that focuses on safety critical components. ISO 26262 is a derivative of IEC 61508.

#### ASILs

ISO 26262 describes five different Automotive Safety Integrity Levels (ASIL) relating to hazard and risk. Ranked from lowest (no) hazard to highest hazard, these levels are: QM, A, B, C and D. A function is assigned an ASIL depending on the severity if the function fails, the probability that the function fails and the controllability of the function, see table 2.1.

Severity	Probability	Controllability		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Table 2.1: ASIL as a function of severity, probability and controllability.

The various integrity levels can be translated into integers (ASIL  $QM = 0$ ;  $A = 1$ ;  $B = 2$ ;  $C = 3$  and  $D = 4$ ). If a hazard requires several components to fail, the added ASIL of these components is used to determine if there is an violation, assuming the components faults are statistically independent of each other. For example, a safety level ASIL B can be met by two independent components which each individually only meet ASIL A (and thus effectively  $A + A = B$ ). [6]

The different ASILs can relate to cost according to various cost heuristics, see table 2.2.

<b>Cost Heuristic</b>	<b>QM</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
Linear	0	10	20	30	40
Logarithmic	0	10	100	1000	10000
Experimental-I [6]	0	10	20	40	50
Experimental-II [6]	0	20	30	45	55

Table 2.2: ASIL cost heuristics.

### **Freedom from interference**

In ISO 26262, Part 1, Definition 1.49, freedom from interference is defined as: Absence of cascading failures between two or more elements that could lead to the violation of a safety requirement. A cascading failure is defined as = "failure of an element of an item causing another element or elements of the same item to fail" (ISO 26262, Part 1, Definition 1.13), and an element is defined as: "system or part of a system including components, hardware, software, hardware parts, and software units" (ISO 26262, Part 1, Definition 1.32)

### **2.2.3 AUTOSAR**

"AUTOSAR (AUTomotive Open System ARchitecture) is a international development partnership of automotive interested parties founded in 2003. It pursues the objective of creating and establishing an open and standardized software architecture for automotive electronic control units (ECUs) excluding infotainment. Goals include the scalability to different vehicle and platform variants, transferability of software, the consideration of availability and safety requirements, a collaboration between various partners, sustainable utilization of natural resources, maintainability throughout the whole "Product Life Cycle"." [5]

The AUTOSAR Architecture distinguishes on the highest abstraction level between three software layers: Application, Runtime Environment (RTE) and Basic Software (BSW) which run on a Microcontroller. [5] See figure 2.2.

- The application software layer is mostly hardware independent.
- The RTE represents the full interface for applications.
- The BSW is divided in three major layers and Complex Drivers: Services, ECU Abstraction and Microcontroller Abstraction. Services are divided furthermore into functional groups representing the infrastructure for System, Memory and Communication Services.

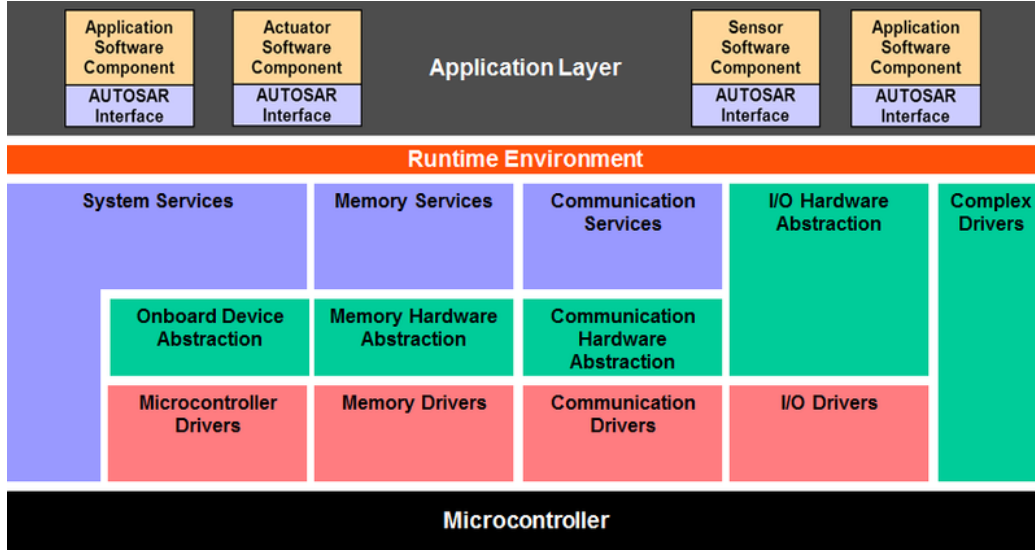


Figure 2.2: AUTOSAR. [5]

## 2.3 Mixed criticality platform solutions

### 2.3.1 Hypervisors

Xen hypervisor Xen zynq SEL4 microkernel Sierra visor SafeG SICS Thin hypervisor

## 2.4 Platooning

Vehicle platooning can be described as a chain of vehicles traveling at a given intermediate distance and velocity. The primary objective for each vehicle with respect to safety is to maintain its distance to the preceding vehicle in the platoon. A platoon of  $N$  vehicles is often modeled in the literature as a set of moving point masses

### 2.4.1 Benefits of platooning

Potential benefits of vehicle platooning includes lower fuel consumption, less road space required and more efficient traffic flow.

Using simulations of platooning, Alam, Gattami, and Johansson [1] showed in 2010 that there is a 4.7–7.7% fuel reduction potential in heavy duty vehicle platooning at a set speed of 70 km/h with two identical trucks.

### 2.4.2 Safety requirements for platooning

System uncertainties or varying vehicle parameters, such as mass, could cause a difference in braking capabilities between the vehicles. Having different braking capacity changes the shape of the safe sets. If the follower vehicle has a higher braking capacity, the level surface derived for neighboring vehicles with identical braking capacity will shift in the positive  $v_{21}$ -direction and the slope of the surface will decrease, as shown in Fig. 4. This means that the follower vehicle will be able to lie closer without endangering a collision. The minimum safe relative distance is therefore shorter compared to the case of two identical vehicles. However, if the lead vehicle has a greater braking capability a perturbation arises in the level surface at  $v_{21} = 0$  and the slope becomes steeper. In this case a minimum distance of  $d = 13$  m must be maintained to remain outside the reachable set at normal mode. Thus, the relative distance must be increased significantly if the lead vehicle has a stronger braking capability. Delays for the platoon control system commonly occur due to detection, transmission, computation, and producing the control command. A delay in the system implies that the lead vehicle will be able to act, change the relative velocity and distance, before the follower vehicle is able to react. A delay can be translated into a shift of the reachable set in Fig. 3 by  $\delta d$  units in the positive direction along the  $d$ -axis and by  $\delta v_{21}$  units in the negative direction along the  $v_{21}$ -axis. However, no change occurs in the follower vehicle's velocity  $v_2$ , since it does not react. Depending on the radar and the collision detection algorithm, a worst-case delay is approximately 500 ms for the considered vehicles. Hence, the lead vehicle will be able to reduce the relative velocity by 3.25 m/s and the relative distance by 0.8 m if it is driving 25 m/s at normal mode. Thus if the follower vehicle maintains  $d \geq 2$  m, a collision can always be avoided for two identical vehicles according to the safe set in Fig. 3. [2]

# Chapter 3

## Current system

This chapter will describe the EMC<sup>2</sup> development platform, for more information see the report by Zaki [28].

### 3.1 Soft overview

The EMC<sup>2</sup> Development Platform (EMC2DP) consists of a Zynq-7000 System on Chip (SoC). Connections. I/O. Bild.

### 3.2 Hardware

The Zynq-7000 SoC has a Processing System (PS) consisting of a hardwired application processing unit, memory controller, and peripheral devices. The main processing unit is a dual-core Cortex-A9 ARM processor. Connected to the PS region is a Programmable Logic (PL) region. The PL is based on Xilinx's 7-series FPGA technology. Due to the flexible nature of the PL, systems can be designed to reach a new level of performance. For example, the PL region can be used to instantiate standard or custom IP hardware modules that can serve as accelerators for the PS. Additionally, the PL region enables the PS to access system resources that are only accessible by the PL. An overview of the interfaces between the PS and the PL can be seen in Figure 3.1.



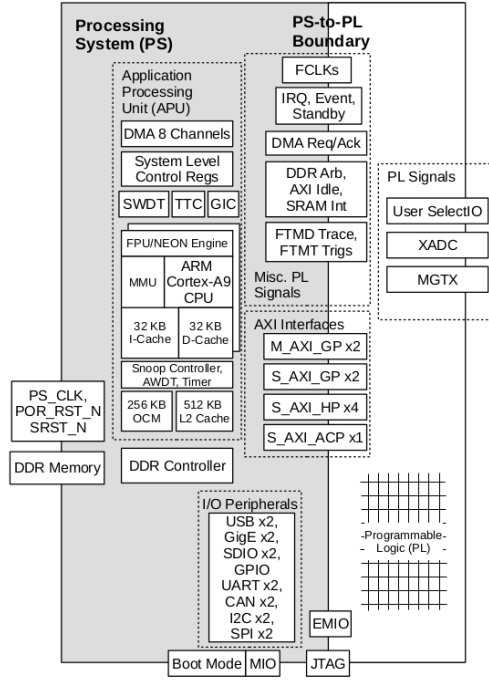


Figure 3.1: Overview of the interfaces between the PS and PL regions. [28]

The interfaces between the PS and PL regions can be divided into two categories:

- Functional interfaces: include the Advanced eXtensible Interface (AXI) ports such as AXIGP for general purpose master/slave device interface between PS and PL regions, extended MIO (EMIO) which enable PL IPs to access most I/O peripherals, interrupts, DMA flow control, clocks, and debug interfaces.
- Configuration interfaces: these signals are connected to the configuration block of the PL, which allow the PS to control the configuration of the PL.

Resources are separated as secure and non-secure using ARM TrustZone [3].

### 3.3 TrustZone

TrustZone is a security feature by ARM that is available in their modern processors. TrustZone creates a security infrastructure in order to protect

critical system assets from being accessed by non-trusted sources. This is achieved by enabling the partitioning of system components, both hardware and software, into either a Secure or Normal zone. Resources that are marked as Normal are not permitted to access Secure resources. This mechanism is enforced by the AMBA3 (Advanced Microcontroller Bus Architecture) AXI bus system. It contains an extra control signal for each of the read and write channels that dictate the access rights Non-secure bus masters to the Secure slaves. Each processor with an enabled TrustZone security extension can be partitioned into a Normal and a Secure virtual CPU. The virtual processors execute in a time-multiplexed fashion, and use the "Monitor Mode" state to create an efficiently switching mechanism between Normal and Secure zones. [28]

### 3.4 Operative systems

The EMC2DP uses two Operative Systems (OS) to create temporal and spatial separation between safety-critical and non-critical applications using TrustZone. In its current setup the Real-Time Operative System (RTOS) FMP by TOPPERS [20] is used for safety-critical applications. This RTOS follows the uITRON4.0 specification [17], which is a widely used RTOS specification for Japanese embedded systems. For non-critical applications, the General Purpose Operative System (GPOS) Linux kernel 4.4 is used. Instead of Linux another instance of FMP could be used for non-critical applications.

### 3.5 Virtual Machine Monitor

A Virtual Machine Monitor (VMM) or "Hypervisor" is used to alternate between the safety-critical (S\_OS) and non-critical (NS\_OS) OS. The VMM used is SafeG [21], also developed by TOPPERS. It switches processor state via a hardware switch. See figure 3.2. The switching takes  $2 \mu s$ .

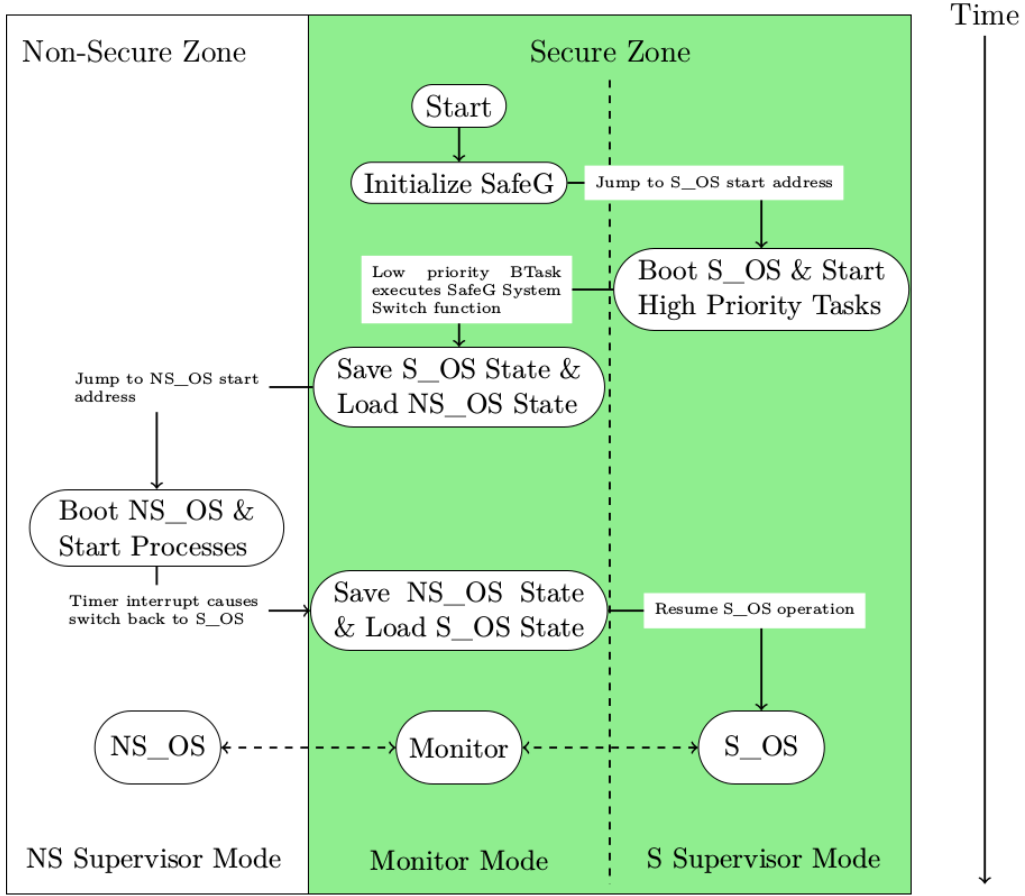


Figure 3.2: Flowchart of the boot sequence of the CPU. [28]

The time it takes the VMM to switch processor state bounds the maximum frequency a task can have while the processor still manages to maintain its switching capabilities. The maximum frequency,  $f_{max}$ , can be calculated as

$$f_{max} = \lim_{e_s, e_{ns} \rightarrow 0} \frac{1}{e_s + e_{ns} + 2e_{switch}} = 250 \text{ kHz}$$

where  $e_s$  is the computational time of the tasks on the S-OS,  $e_{ns}$  is the computational time of the tasks on the NS-OS and  $e_{switch}$  is the time required for the mode-switch.

An basic overview of the hardware and the software of the system can be seen in Figure 3.3.

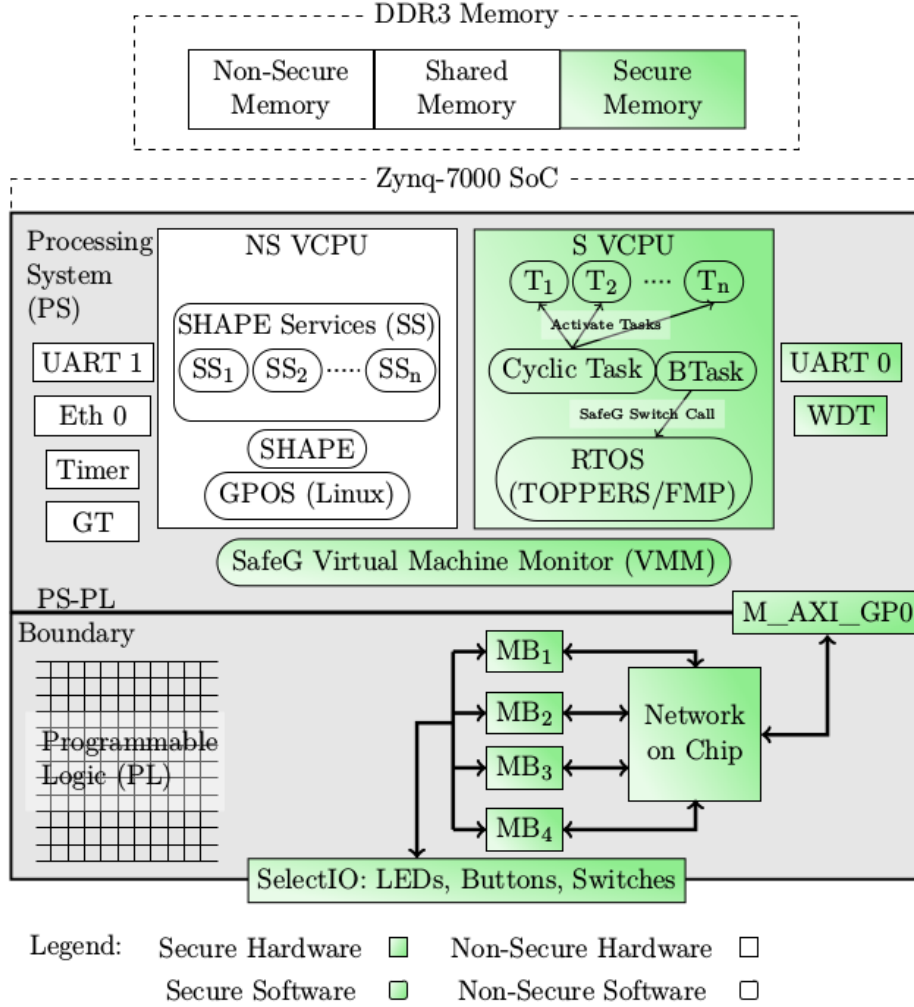


Figure 3.3: Overview of the MCS in place. [28]

### 3.6 Build procedure

The MCS is built from many different components. Hardware design, applications, virtualization layer, operative systems, boot loaders etc. This section will describe the build procedure.

Xilinx’s software Vivado [26] is used to synthesize the hardware design (vhdl or verilog code) into a bitstream file (.bit) in order to configure the PL region of the Zynq. Vivado also produces a set of files that represent the

designed hardware platform, which are used for software development. Xilinx SDK tool is used to create the Board Support Package (BSP) and the First Stage Boot Loader (FSBL) that correspond to the designed system. In general, after the FSBL initialization process completes, and depending on boot sequence, the CPU can do any of the following actions: configure the FPGA, initiate the Second Stage Boot Loader SSBL, or jumps to the first address of the main program. The SDK tool is also used to generate a boot file (BOOT.bin), which must at least contain the FSBL (fsbl.elf). In the implemented system, the BOOT.bin file also includes the bitstream file (system.bit) and the SSBL (uboot.elf \* ). Once the system is initialized and the PL is configured, the system starts executing the u-boot instructions present in the BOOT.bin. U-boot is a full system on its own, and has many useful features. In particular, u-boot can be used to load executables and other system files from a remote server into the DDR3 memory using protocols such as Trivial File Transfer Protocol (TFTP), see Figure 3.4.

Figure 3.4 provides a summary of the different dependencies for the system and the required flow for building the system. The keyword "step x" indicates instances where dependencies exists within a build directory. Software tools are indicated by the circular shape, such as Vivado, SDK, and GNU Compiler Collection (GCC) (make).

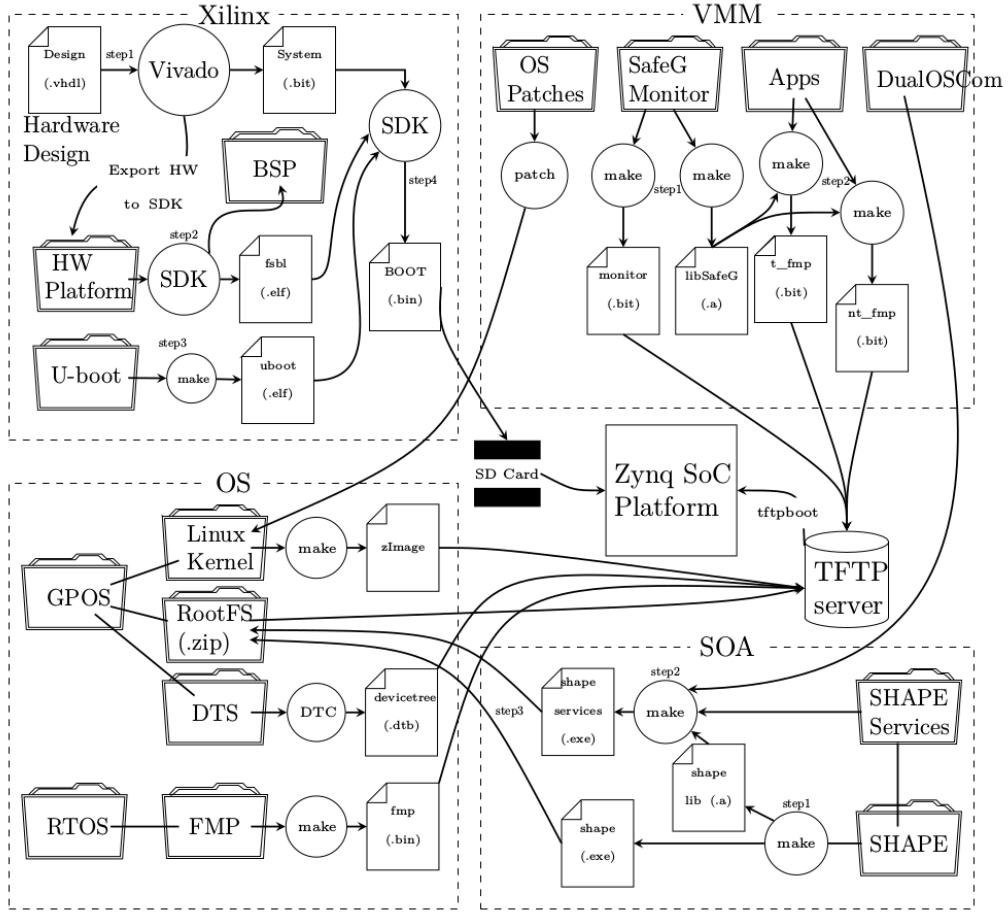


Figure 3.4: System build procedure. [28]

For more information about the build and the system, see the report by Zaki [28].

# Chapter 4

## System design

This chapter will derive the design of the controller to be implemented.

# Chapter 5

## Implementation

This chapter will describe the implementation of the control system in the demonstrator.



# Chapter 6

## Results

This chapter will present results from the demonstrator to the reader.

# Chapter 7

## Discussion

Discussion about the results produced by the thesis.

### 7.1 Information retrieval

Printing to SYSLOG takes time and affects the system that's being monitored.

### 7.2 Utilization

Deterministic system - work towards 100%, anything under that: reduce clock frequency to reduce power consumption and reach 100%.

Sporadic system - probably want to be around 50% utilization to maintain 100% schedulability, higher depending on requirements on performance versus requirements on efficiency.

# Chapter 8

## Future work

This chapter will contain thoughts and ideas for future work building on this thesis or in the area of MCS in general.

### 8.1 MCS using virtualization

Facilitate for more than two different criticality levels.  
Examine different scheduling methods.

### 8.2 MCS using other means of partitioning

Examine limitations for other configurations of MCS, for example different CPUs for different criticality levels.

### 8.3 Amount of criticality levels

Research should be done to investigate how many different levels of criticality,  $n$ , to facilitate for on MCS in different industries. In the automotive for example,  $n$  should be between 1 and 5 since ISO26262 defines 5 different ASILs. If the applications in a car are spread uniformly across all criticality levels it might be of higher interest to have  $n$  closer to 5. Similarly, if the applications are heavily concentrated on a certain criticality level,  $n$  probably should be closer to 2.

## 8.4 Economical benefits for pursuing MCS

It is not clear how much the potential economical benefit would be from pursuing MCS. The economical impacts of MCS might be different in different industries. It must be calculated more exactly how large the potential benefits would be to gauge the need for pursuing MCS.

# Bibliography

- [1] Assad Alam, Ather Gattami, and Karl H. Johansson. An experimental study on the fuel reduction potential of heavy duty vehicle platooning. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 306–311, September 2010.
- [2] Assad Alam, Ather Gattami, Karl H. Johansson, and Claire J. Tomlin. Guaranteeing safety for heavy duty vehicle platooning: Safe set computations and experimental evaluations. *Control Engineering Practice*, November 2013.
- [3] ARM Ltd. ARM TrustZone, February 2017. <https://www.arm.com/products/security-on-arm/trustzone>.
- [4] N C Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 5 2001.
- [5] AUTOSAR. AUTOSAR Homepage, February 2017. <http://www.autosar.org/>.
- [6] Luís Silva Azevedo, David Parker, Yiannis Papadopoulos, Martin Walker, Ioannis Sorokos, and Rui Esteves Araújo. *Exploring the Impact of Different Cost Heuristics in the Allocation of Safety Integrity Levels*, pages 70–81. Springer International Publishing, Cham, 2014.
- [7] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43, Nov 2011.
- [8] Sanjoy Baruah and Steve Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications, 2008.
- [9] Carl Bergenhem, Henrik Pettersson, Erik Coelingh, Cristofer Englund, Steven Shladover, and Sadayuki Tsugawa. Overview of platooning systems. In *Proceedings of the 19th ITS World Congress*, Vienna, Austria, October 2012.

- [10] Alan Burns and Robert I. Davis. Mixed criticality systems - a review. Available online: <https://www-users.cs.york.ac.uk/burns/review.pdf>, July 2016.
- [11] Dictionary.com. safety-critical system, January 2017. <http://www.dictionary.com/browse/safety-critical-system>.
- [12] Software considerations in airborne systems and equipment certification. Standard, Radio Technical Commission for Aeronautics, Washington, DC, USA, December 2011.
- [13] Encyclopedia.com. safety-critical system, January 2017. <http://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/safety-critical-system>.
- [14] Anne Håkansson. Portal of research methods and methodologies for research projects and degree projects, July 2013.
- [15] Functional safety of electrical/electronic/programmable electronic safety-related systems – parts 1 to 7. Standard, International Electrotechnical Commission, Geneva, CH, April 2010.
- [16] Road vehicles – functional safety – part 9: Automotive safety integrity level (asil)-oriented and safety-oriented analyses. Standard, International Organization for Standardization, Geneva, CH, November 2011.
- [17] ITRON Committee, TRON Association. uITRON4.0 Specification Ver. 4.00.00. <http://www.ertl.jp/ITRON/SPEC/FILE/mitron-400e.pdf>.
- [18] Max Lemke et al. Mixed criticality systems. report from the workshop on mixed criticality systems, February 2012.
- [19] SafeCOP. SafeCOP - part B, October 2016.
- [20] TOPPERS Project, Inc. Introduction to the TOPPERS/FMP kernel, February 2017. <https://www.toppers.jp/en/fmp-kernel.html>.
- [21] TOPPERS Project, Inc. TOPPERS SafeG, February 2017. <https://www.toppers.jp/en/safeg.html>.
- [22] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, 2007.

- [23] W. Weber, A. Hoess, F. Oppenheimer, B. Koppenhöfer, B. Vissers, and B. Nordmoen. EMC2 a Platform Project on Embedded Microcontrollers in Applications of Mobility, Industry and the Internet of Things. In *2015 Euromicro Conference on Digital System Design*, pages 125–130, August 2015.
- [24] Werner Weber. A Platform Project on Embedded Microcontrollers in Applications of Mobility, Industry and the Internet of Things, Mars 2015. <https://artemis-ia.eu/publication/download/1131.pdf>.
- [25] Wikipedia.com. Life-critical system, January 2017. [https://en.wikipedia.org/wiki/Life-critical\\_system](https://en.wikipedia.org/wiki/Life-critical_system).
- [26] Xilinx Inc. Vivado Design Suite, March 2017. <https://www.xilinx.com/products/design-tools/vivado.html>.
- [27] Xilinx Inc. Zynq-7000 All Programmable SoC, February 2017. <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>.
- [28] Youssef Zaki. An embedded multi-core platform for mixed-criticality systems: Study and analysis of virtualization techniques. Master’s thesis, KTH, School of Information and Communication Technology (ICT), 2016.