Safety-critical Control in Mixed Criticality Embedded Systems

An evaluation of the Alten MCS used in vehicle platooning

Master Thesis
Royal Institute of Technology
Stockholm, Sweden

Emil Hjelm
`emilhje@kth.se`

June 9, 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 efficiency it is desired to combine multiple applications into fewer ECUs, leading to mixed criticality embedded systems. The assurance of safety critical applications not being affected by non-critical applications on the same system is crucial.

Sammanfattning

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

Preface

Credit where credit is due.

<div align="right">

Emil Hjelm
Stockholm

</div>

Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Description
ECU	Electronic Control Unit
MCS	Mixed Criticality System
EMC^2	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^{2}DP	EMC^2 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^2" to the reader.

1.1 Background

Today, modern automotive systems contain around 70-100 Electric Control Units (ECUs) [25]. Each ECU controls a subsystem of a specific criticality level such as safety-critical anti-lock brake system, or non-critical entertainment systems [37]. 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 hardware platform, leading to a Mixed Criticality System (MCS). However, this approach increases system complexity, and hinders the certification of safety-critical systems [43]. To enable design, test, and certification of MCS, spatial and temporal partitioning can be used in the architecture of the system.

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].

With cars being increasingly connected to the internet, security poses another concern for MCS. If one part of a system is easier to attack, the entire system could be compromised if proper isolation is not in place.

The EU project "Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments" (EMC^2) 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" [37].

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 [15] it is defined as "A system in which any failure or design error has the potential to lead to loss of life." Further, [12] defines safety-critical systems as "A computer, electronic or electromechanical system whose failure may cause injury or death to human beings." A Wikipedia article, [39], 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 [18] defines four different levels and the ISO 26262 standard [21] and the DO-178C standard [14] 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 Alten Mixed Criticality System

As a part of the EMC^2 project, Alten has developed a system for handling applications of mixed criticality on the same piece of hardware, Alten MCS. The MCS developed at Alten has been implemented on two development boards, the Zedboard (Zynq Evaluation and Development board) and the EMC^2 Development Platform, or EMC^{2}DP. Both boards have a Xilinx Zynq-7000 SoC [42]. Alten MCS 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 hypervisor is used to alternate between the two. For more information, see chapter 3.

1.1.4 Platooning

Platooning is the concept of a collection of vehicles that travel coordinated in formation in order to reduce air drag. Some expected advantages of platooning include increased fuel and traffic efficiency, safety and driver comfort [9]. Platooning puts high requirements on the control systems in the platooning vehicles, since failure of these systems could have fatal consequences.

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 switches Operative System (OS) via a hypervisor to enable partitioning between safety-critical and non-critical applications, which takes about 2 μs. This means that at higher switching frequencies the mode switch takes up more processor time, potentially making up almost all processor time at very high frequencies.

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. 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 should never crash.

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 [29]. 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 [43].

The embedded computer is constrained to the Xilinx Zynq-7000 SoC [42]. The hypervisor used is constrained to SafeG [35] and the RTOS is constrained to FMP [34].

1.6 Research design

The plan is to conduct a confirmatory investigation using quantitative data/operations with a deductive approach. This is a quantitative research method. [17].

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, as this could have catastrophic consequences. Because of this the requirement of partitioning must have higher priority than the need of sharing resources.

Chapter 2

State of the art

This chapter will go through relevant articles and research 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 into a MCS is higher resource efficiency, which potentially 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 EMC² project aims at creating platforms for easier development of MCS.

The EMC^2 project lists several goals [38]:

- 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 Security concerns

Today, cars are increasingly becoming networked IT entities. With increased connection, security threats are also increasing, and the most visible gateway for hacking attempts in cars is through the infotainment systems. This gives additional incentive to ensure proper isolation in MCS.

2.1.3 Sharing processor

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

2.1.3.1 Conventional scheduling

The simplest scheduling algorithm is First In First Out (FIFO). As the name suggests, the first task to enter the queue is also executed first, and executes until it is finished [4].

Another simple scheduling algorithm is Fixed Priority (FP). Each task is assigned a priority level, and the task with the highest priority in the queue is allowed to execute [26].

The scheduling algorithm Rate Monotonic (RM) assigns a priority level to a task depending on its frequency. Higher frequency leads to higher priority. The algorithm was proven optimal under the assumption that the deadline of every task coincided with its rate [26].

Deadline Monotonic (DM) assigns a priority level to a task depending on its deadline. Shorter deadline leads to higher priority [26].

Earliest Deadline First (EDF) is a dynamic scheduling algorithm that executes the task with the least amount of time left until its deadline.

Round Robin (RR) scheduling lets each task in the job queue execute for a predefined amount of time (called a time quanta), and if a task does not complete it is preempted and placed back in the queue waiting for its next time slot [23].

2.1.3.2 Mixed criticality scheduling

The area of sharing the processor in MCS was first explored by Steve Vestal [36] in 2007. His paper showed that neither Rate Monotonic (RM) nor Deadline Monotonic (DM) priority assignment was optimal for MCS.
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 [36]. 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.
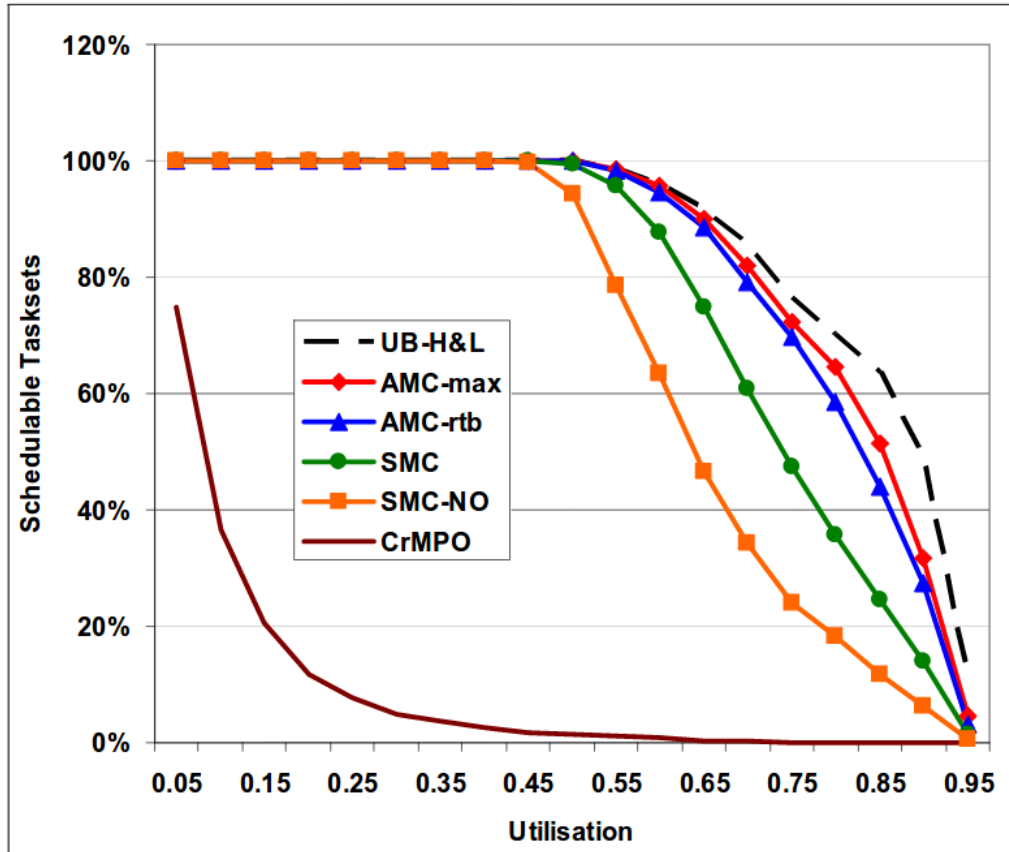
Figure 2.1: Percentage of schedulable tasks. [7]

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

2.1.3.3 Execution times

A key concept when designing a real time system is the worst case execution time of each task, or WCET. This is the longest amount of time a task will take to execute on the specific hardware. The scheduling in a RTOS is designed so that every task is guaranteed to execute until it is finished. To be able to guarantee this, the WCET of each task must be known. There are two approaches to learn the WCET of a task, static and measurement-based methods [40].

Static based methods involve analysis of code, counting assembler instructions for each part of the code and combining these to get a maximum amount of instructions, where each instruction has a known execution time [40].

Measurement based methods are a bit more straight-forward, you simply let the code execute many times and measure the time each execution of the code takes. This can be measured by for example setting a output pin high at the start of the task and then setting it low at the end, and have an external clock measure the time the pin is high [40].

2.2 Standards

Safety practices are becoming more regulated as industries adopt a standardized set of practices for designing and testing products. To ensure safe and secure practices in industries, safety standards are becoming more regulated. Different standards address different areas. Below, the two safety standards IEC 61508 and ISO 26262 will be described.

2.2.1 IEC 61508

IEC 61508 [18] 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 [21] addresses the needs for an automotive-specific international standard that focuses on safety critical components. ISO 26262 is a derivative of IEC 61508.

2.2.2.1 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.

Cost Heuristic	QM	A	B	C	D
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.

2.2.2.2 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 partnership between several automotive interested parties. The partnership aims at establishing and standardizing the software architecture for automotive ECUs, excluding infotainment [5]. Contributors include BMW, Daimler, Ford, GM, Toyota and Volkswagen.

2.3   Hypervisors

A hypervisor (or virtual machine monitor) is a piece of software capable of running several operating systems on the same hardware system. The hypervisor can be seen as an interface between the operative systems and the hardware of a system. In order to facilitate for multiple operative systems of different criticality an appropriate hypervisor should be used. A system will only be as secure as its hypervisor, so it is important that the hypervisor has been engineered to at least the same standards as the most critical functions that will be implemented on the system. Other things to consider is that some hypervisors can accommodate multiple instances of operative systems, and some only facilitate for two. A description of a few available open-source hypervisors will follow.

2.3.1   SafeG

SafeG (short for *Safety Gate*) is a hypervisor developed by the TOPPERS group of Nagoya University in Japan [35]. It can host two operating systems on the same hardware, partitioning them into Trusted and Non-Trusted zones (sometimes called Trusted and Non-Trusted "worlds") via ARM Trust-Zone [3].

SafeG guarantees both that the memory and temporal execution of tasks in the OS on the secure side is isolated from the OS on the non-secure side [30]. Both memory and time isolation is backed by the ARM TrustZone hardware extensions [30]. RTOS tasks are triggered by FIQ (Fast Interrupt Request),

an interrupt with higher priority than IRQ (Interrupt Request), which is allocated to the GPOS [30].

Figure 2.2 depicts the SafeG architecture. It shows a simplified view of a TrustZone enabled ARM processor together with partitioned memory and device IO.
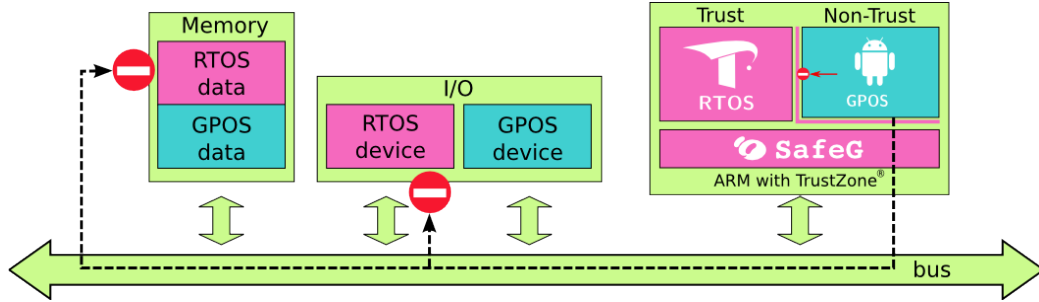

Figure 2.2: SafeG and TrustZone.

The features of SafeG as described on their website are as follows [35]:

- Enables the concurrent execution of RTOS and GPOS on either single-core or multi-core ARM-based platforms.

- Devices and memory regions that are configured as Secure are protected against illegal GPOS accesses.

- Normal world devices can be accessed from both GPOS (Non-Trusted) and RTOS (Trusted) software.

- Real-time requirements are guaranteed in RTOS (Trusted) via the utilization of FIQ and IRQ interrupts, where FIQ interrupts are issued for RTOS and IRQ interrupts are issued for GPOS. While in the Trusted state, IRQ interrupts are disabled so that GPOS can not disturb the execution of the RTOS. Therefore, GPOS only executes when the RTOS issues the Secure Monitor Call (SMC) instruction, which causes the SafeG monitor to switch from the Trusted world (RTOS) to the Non-Trusted world (GPOS). Furthermore, FIQ interrupts are active during the execution of GPOS, which enables RTOS to retake control of the system. For example, a cyclic execution of RTOS/GPOS can be controlled by an FIQ interrupt of a system timer.

- GPOS does not require any major changes, and can execute with minimal overhead.

13

- Includes an efficient guest-to-guest communication mechanism (i.e.referred to as SafeG COM).

The RTOS tasks are statically mapped to a processor during compilation, while the GPOS uses all available virtual CPUs in Symmetric Multi-Processor mode.

The SafeG Monitor is the gateway between the GPOS and the RTOS. During the switch operation, it is responsible for saving the state of one zone and loading the state of the other. According to TOPPERS [31], the WCET of switching the OS is just under $2\mu s$, see table 2.3.

Switch path	WCET
Switch from RTOS to GPOS	$1.5\mu s$
Switch from GPOS to RTOS	$1.7\mu s$

Table 2.3: WCET of switching OS. [31]

2.3.2 seL4 microkernel

seL4 is a microkernel developed, maintained and formally verified by NICTA (now the Trustworthy Systems Group at Data61) and owned by General Dynamics C4 Systems [28]. seL4 is formally verified, which implies that its specification is verified mathematically [11]. The kernel follows the "minimality principle", meaning that it "allows features in the kernel only if the required functionality could not be achieved by a user-level implementation" [27]. As a result, the microkernel is small, efficient, and robust.

Scheduling in seL4 is deliberately left underspecified. The present implementation uses a fixed-priority round-robin scheduler [27].

2.3.3 SICS Thin hypervisor

SICS Thin Hypervisor (STH) is a light-weight hypervisor designed for ARM-based devices [13]. STH runs directly on top of the hardware (bare metal), and achieves system virtualization through paravirtualization. STH allows for more than two guests to run on top of the hypervisor. STH has no Trust-Zone support.

The current version of STH supports ARMv5 (926EJ-S) and ARMv7 Cortex-A8 only, but due to its flexible nature and minimal layer of hardware abstraction it is easily migrated to other platform [13]. There is no graphical support

and all communication with the kernel is done through the UART.

2.3.4 Sierra visor

SierraVisor is a bare metal universal hypervisor created by Sierraware [33]. It supports paravirtualization, TrustZone virtualization, and hardware assisted virtualization. The SierraVisor Hypervisor supports any ARM11, Cortex-A9, or Cortex-A15 platform, but only Cortex-A15 supports the hardware assisted virtualization option.

2.4 Field Programmable Gate Array

A Field Programmable Gate Array (FPGA) is an array of logic gates with reconfigurable interconnects that can be connected to each other. As the name suggests, it is possible to reprogram the physical function of the FPGA "in the field". This is what distinguishes a FPGA from an Application Specific Integrated Circuit (ASIC), an ASIC can not be reprogrammed after manufacturing.

To program the behavior of the FPGA, Hardware Descriptive Language (HDL) is used. The two most widely used HDLs are Verilog and VHDL (VHSIC Hardware Descriptive Language, where VHSIC is an acronym for Very High Speed Integrated Circuit) []. Verilog came to existence late 1983. The language was standardized as IEEE standard 1364-1995, and has since been revised and included in IEC/IEEE Behavioural Languages standard 61691 [20]. Work with VHDL formally began in 1981, and in 1987 it was standardized as IEEE-1076. The active standard for VHDL is also part of the IEC/IEEE Behavioural Languages standard 61691 [19]. Using Google Trends [16] with the search terms "vhdl" and "verilog", one can see that Verilog seems to be the more popular language of the two in the USA, India and South Korea, while VHDL is more used in Europe and South America, see Figure 2.3.
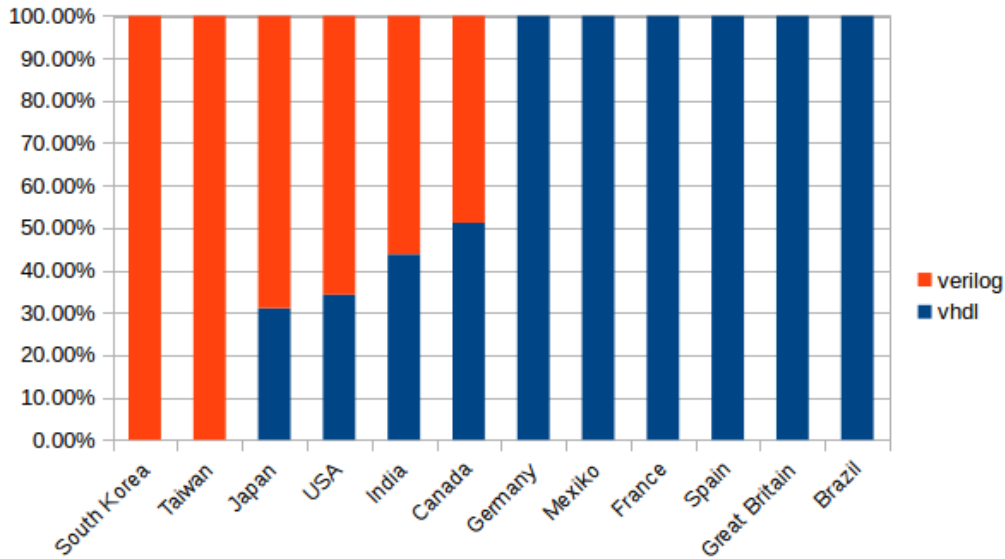
Figure 2.3: Distribution of Google searches for "verilog" vs "vhdl" in the world.

A key difference between code running on a processor and code describing functions in a FPGA is that traditional code is executed in serial, where only one computation can be executed at any one time. In a FPGA however, functions can run in parallel. This makes FPGAs excellent for implementation of hardware functions that require high speed and high frequency that you do not want to take up precious processor time.

Some FPGA circuits have non-volatile memory from where the hardware descriptive language is loaded when the circuit is powered up, this means that its configuration is not lost if its power goes down.

2.5 Platooning

Platooning, road trains or convoy driving is the concept of a chain of vehicles with no physical connection traveling at a given (short) intermediate distance in order to utilize the reduced air friction behind the vehicle in front. The primary objective for each vehicle with respect to safety is to maintain its distance to the preceding vehicle in the platoon.

2.5.1 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.

In 2014, Lammert et. al. [24] showed that platooning can result in reduced fuel consumption of up to 5.3% for the lead vehicle and up to 9.7% for following vehicles.

Tests by truck manufacturer Scania have shown that platooning can reduce fuel consumption by up to 12% [32].

2.5.2 Safety requirements for platooning

With shorter distance between vehicles, the margin of error also decreases. This puts high requirements on the systems controlling the speed of the vehicles in the platoon.

In an article by Alam et al. [2], safe sets for heavy duty vehicle platooning are computed to calculate minimum distance between the vehicles in a platoon without endangering safety. Varying parameters are concidered such as mass, air resistance and road gradient since these could cause a difference in braking capabilities between the vehicles. If the follower vehicle has a higher braking capacity, it 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. 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. Depending on the radar and the collision detection algorithm, a worst-case delay is approximately 500 ms for the considered vehicles [2].

Chapter 3

Alten MCS

This chapter will describe the Alten MCS, its different components and how it is built.

3.1 Overview

The Alten MCS (Alten Mixed Criticality System) is a Mixed Criticality System consisting of both hardware and software components. The general idea is that it should be capable of running two operating systems of different criticality on the same hardware, separated via a hypervisor, where errors from the non-critical OS should not be able to propagate into the safety-critical OS. The Alten MCS has currently been built on two different development boards, the EMC^2 Development Platform (EMC^{2}DP) and the Zynq Evaluation and Development board (Zedboard). Both boards are equipped with a Zynq-7000 System on Chip (SoC). The development boards can be seen in Figure 3.1.
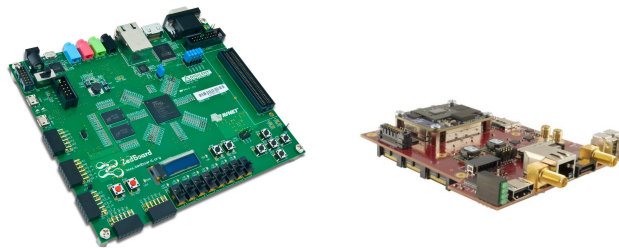
(a) Zedboard   (b) EMC2DP

Figure 3.1

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.

Both the PS and PL regions have support for ARM TrustZone [3].

3.3 Operative systems

The Alten MCS 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 [34] is used for safety-critical applications. This RTOS follows the uITRON4.0 specification [22], 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.

Tasks in FMP are statically allocated to a processor core, defined in a .cfg file. Linux divides its workload across all available processor cores in SMP (Symmetric Multi-Processor) mode.

3.4 Hypervisor

A hypervisor (also known as a Virtual Machine Monitor (VMM)) is used to alternate between the safety-critical (S_OS) and non-critical (NS_OS) OS. The hypervisor used is SafeG [35], also developed by TOPPERS. It switches processor state via a hardware switch. See figure 3.2. The switching takes roughly 2 μs [31].
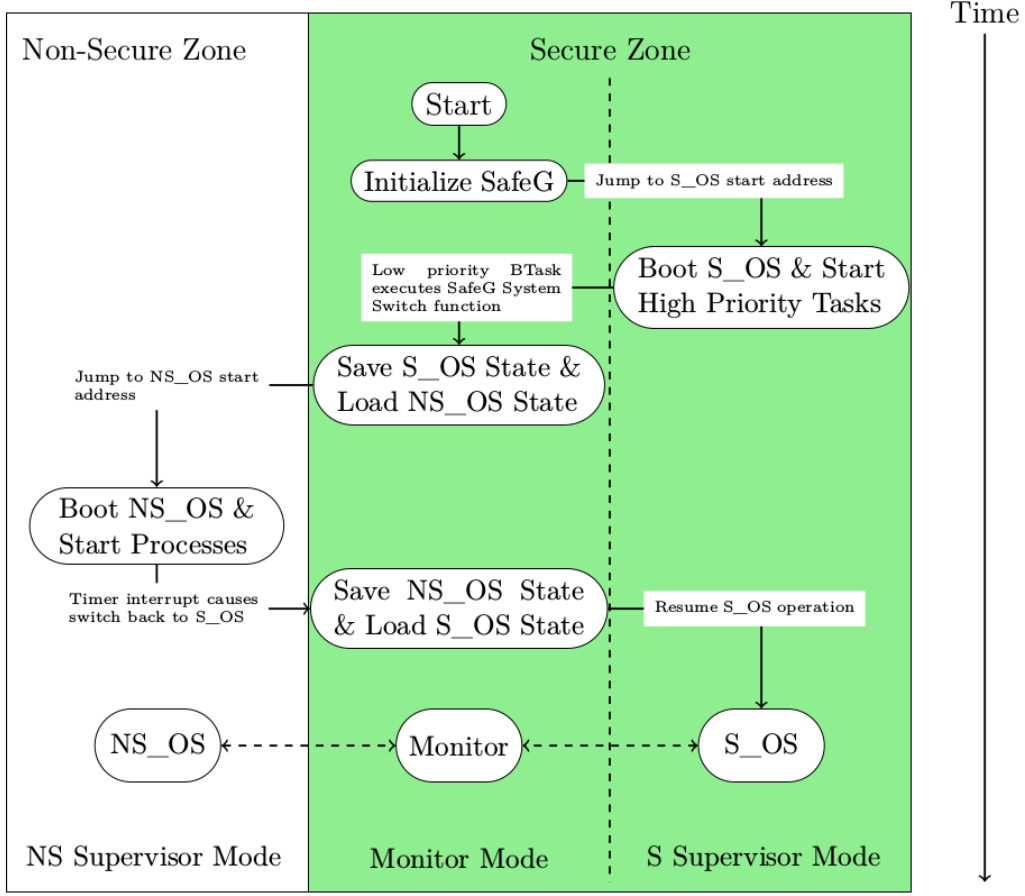
Figure 3.2: Flowchart of the boot sequence of the CPU. [43]

When executing the tasks via the hypervisor, all S_OS tasks are executed according to the scheduling. At the lowest priority is the task that switches processor state. As soon as an S_OS task needs to execute, a FIQ is issued to interrupt the processor and switch mode back to the S_OS.

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

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

where e_s is the execution time of the tasks on the S_OS, e_{ns} is the execution time of the tasks on the NS_OS and e_{switch} is the time required for the mode-

switch.

A more practical limitation is that the periodicity of a cyclic task in FMP is defined by an integer in milliseconds, meaning that the highest frequency for a periodic task is 1 kHz.

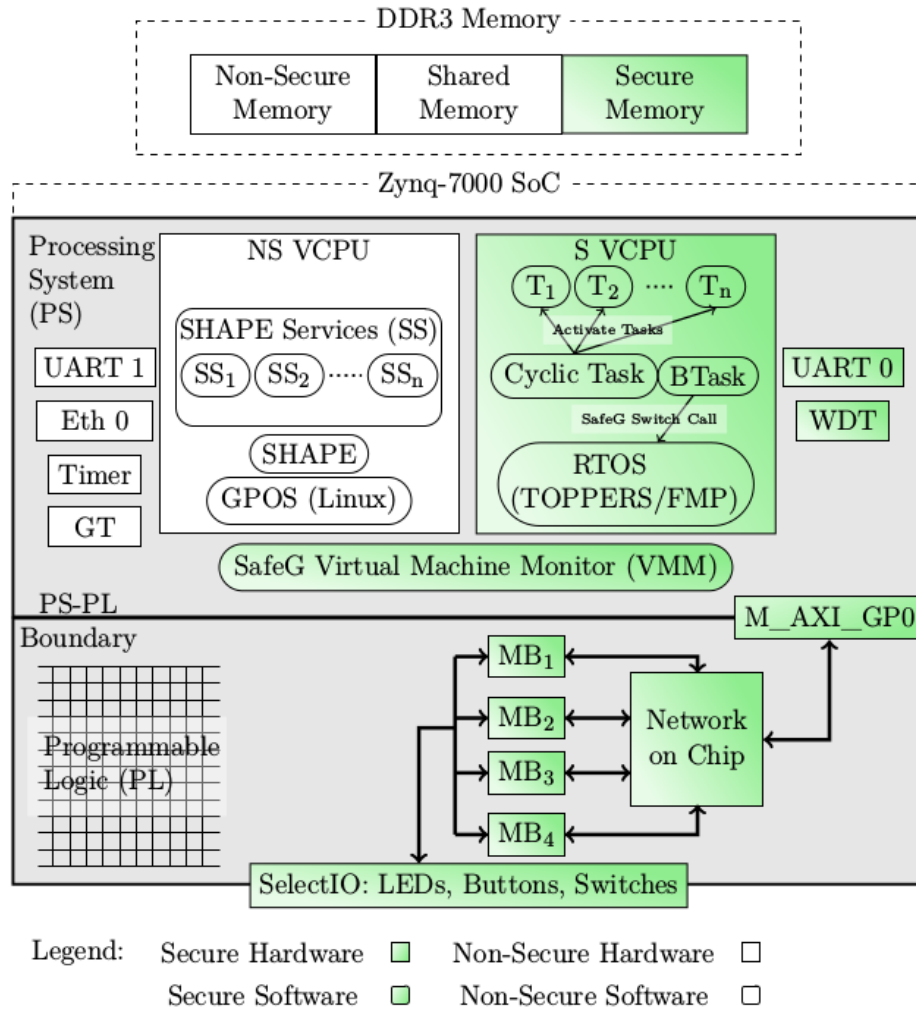A basic overview of the hardware and the software of the system can be seen in Figure 3.3.

Figure 3.3: Overview of the MCS in place. [43]

3.5 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 HLS [41] is used to design the FPGA of the Zynq-7000. In Vivado the different IPs, their interconnect and the Processing System is configured. This results in a bitstream file that is used in Xilinx SDK to generate a Board Support Package (BSP) and the First Stage Boot Loader (FSBL). The BSP contains libraries for functions and variables necessary for the IPs on the FPGA, and the FSBL configures the FPGA correctly. The FSBL is included in the boot file (BOOT.bin) generated in Xilinx SDK. The boot file also contains the Second Stage Boot Loader (SSBL), u-boot. U-boot is a program that can load executables and other system files from a remote server into the DDR3 memory. This is used to load relevant files for the OS into the Zynq, which brings us to the software part of the build procedure. The software is divided into three parts: RTOS, GPOS and monitor. The SafeG monitor is built and compiled using GCC (Gnu Compiler Collection), resulting in monitor.bin. The RTOS FMP is built and compiled from a .cfg file where tasks and handlers are created, a .c file where tasks and handlers are defined and a .h file containing declarations. This results in fmp.bin. For the non-secure side either Linux or another instance of FMP could be used. The process for FMP is the same as with the secure side. For Linux on the non-secure side, the Linux kernel needs to be modified to support SafeG. This results in the files zImage, devicetree.dtb. All of these files are created and stored on a server and are then loaded into the Zynq-7000 using u-boot.

Figure 3.4 provides a summary of the different dependencies for the system and the required flow for building the system.
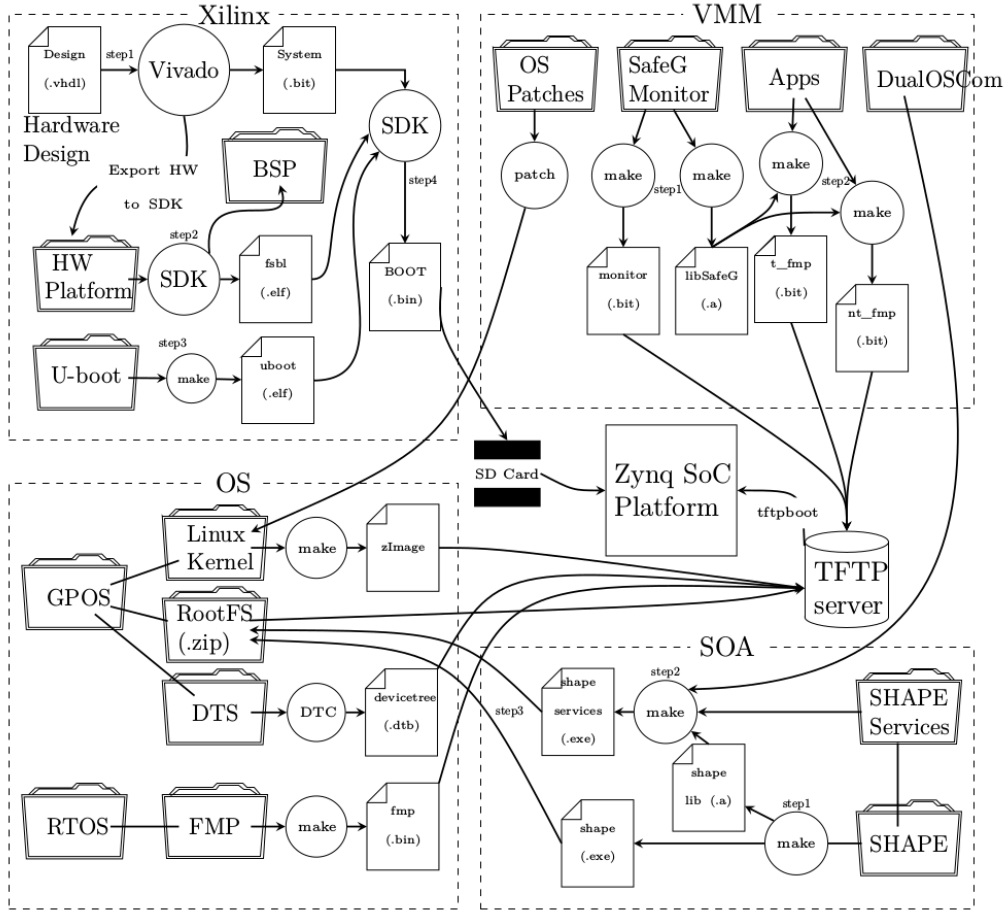
Figure 3.4: System build procedure. [43]

This build procedure setup provides for easy modifications to a small part of the system without having to rebuild other parts, and the board to which the code should be loaded into only needs to be connected to a network socket and a laptop.

Chapter 4

System design

For the system to accomplish the goals stated in 1.2, several hardware and software functions need to be designed and implemented.

4.1 Operative system functions

This section will describe the functions to be implemented in the RTOS.

4.1.1 Longitudinal control

A PID controller will be implemented to control the speed of the vehicle and its distance to the vehicle in front of it. The controller will use information from the preceding vehicle and a distance sensor as input to calculate a control signal.

CC-ACC-CACC, architecture, shared info.

For more information, see the report by Roshanghias [?].

4.1.2 Lateral control

A lane-detecting algorithm will be implemented on a Raspberry Pi that will send data to the Alten MCS. The choice of Raspberry Pi was made because of the amount of open source video processing code available for the platform. For more info, see the report by Ferhatovic [?].

Lateral control will be implemented on the RTOS on the Alten MCS. The function receives the deviation from the center line as calculated by the lane

detection algorithm from the Raspberry Pi, and calculates a control signal from this.

4.1.3 Data aggregation

This function will read wheel sensor data and calculate if the road conditions are unsafe for platooning. See the report by Hellman [**?**].
The function will also serve as a filter for distance measurements and speed measurements for other tasks to use.

4.1.4 Communication

To maintain secure communication between the two vehicles in the platoon, a communication protocol will be developed. WiFi messages will be parsed in a MicroBlaze processor in the FPGA and sent to the OS. For more information, see the report by [**?**].

4.2 Hardware implemented functions

This section will describe the functions to be implemented on the FPGA, also known as IPs.

4.2.1 Speed reading

To read the speed of the vehicle, encoders need to be connected to the wheels. The four encoders will send pulses very quickly and a hardware decoder is needed to process them and send the speed of each wheel to the OS.

4.2.2 Distance reading

To read the distance to the preceding vehicle, a LIDAR (LIght Detection And Ranging) will be used. The LIDAR communicates the distance it is reading via a PWM signal. A hardware function is needed to read the PWM pulse and convert it to a distance.

4.2.3 Communication

WiFi communication is used between the two vehicles in the platoon. To read packages, a WiFi module is used that communicates via UART with the Alten MCS. A MicroBlaze in the FPGA processes the packages and sends

the parsed data to the OS via a mailbox. For more information, see the report by [?].

4.3 Overview design

An overview of the different functions to be implemented on the Zynq-7000 can be seen in figure 4.1.
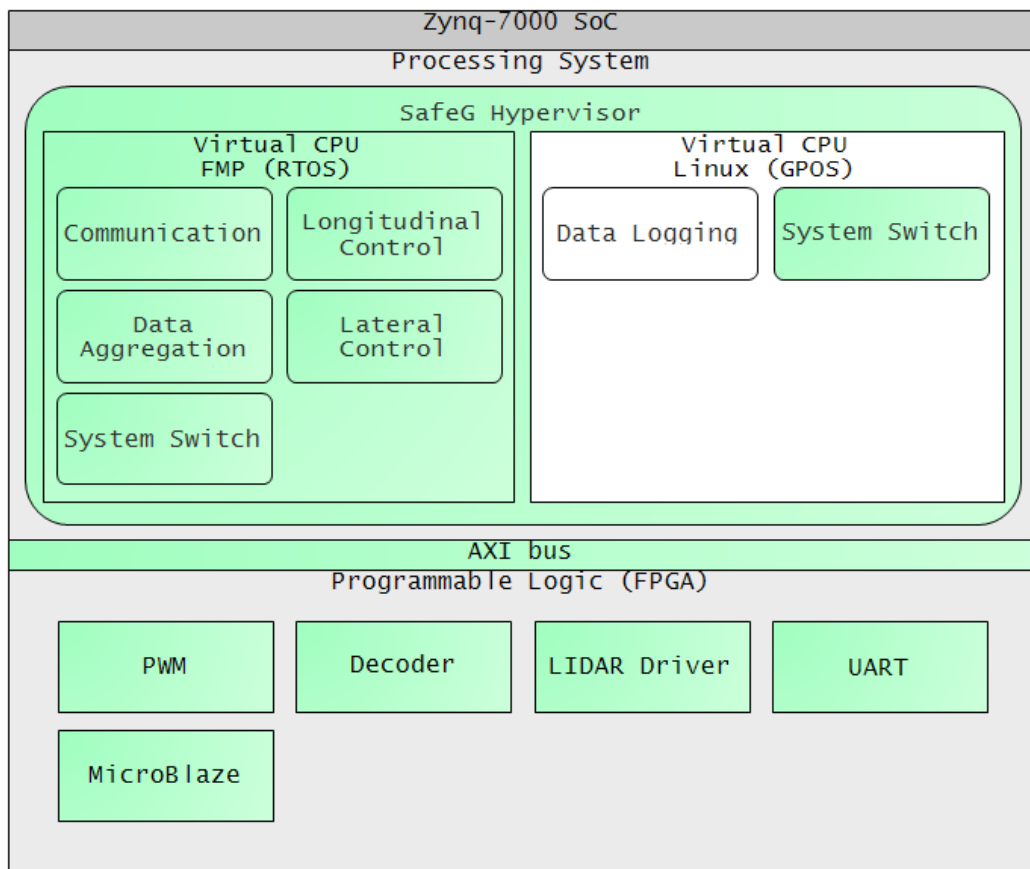

Figure 4.1: Overview of the different functions to be implemented.

A sequence diagram of the various dependencies and flow of each function can be seen in figure 4.2.

Figure 4.2: Sequence diagram of the various hardware and software functions.

The Vivado block design can be seen in Appendix A.

Chapter 5

Implementation

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

5.1 Platform

To demonstrate the system, the RC car UTOR 8E from BSD racing was chosen. It has a steering servo, brushless DC motor,
PWM signal to driver for motor and servo, 50Hz. The speed is controlled by the pulselength of the PWM signal. This was tested with the handcontroller to see what PWM signals were received at the end points. Maximum and minimum high time measured was 0.02 seconds and 0.01 seconds. Depending on how high frequencies the driver can handle, the frequency could be increased without changing the high time, potentially up until 0.02 seconds would correspond to a duty cycle of 100%, see figure.
This was tested, and gave expected results for frequencies up to 200 Hz. After this

5.2 Electrical schematics

Two 7.4V batteries in parallel. Voltage regulator to 5V for LIDAR, Raspberry Pi, encoders and WiFi module. Voltage regulator to DC input voltage for Zedboard.

5.3 Hardware functions

5.3.1 Pulse Width Modulation

To control the speed of the motor and the position of the steering servo, Pulse Width Modulation (PWM) is used. Two hardware implemented PWM functions will be needed.

5.3.2 LIDAR

To read the distance to the preceding vehicle, the LIDAR Garmin LIDAR Lite v3 was used. Specs. The PWM interface was used. A hardware function was implemented on the FPGA to read the pulse width from the PWM signal. The length of the pulse width was counted and written on an AXI bus address, ready for the OS to be read and convert into a distance in cm. The hardware code was written in VHDL and can be seen in Appendix B.

5.3.3 Encoders/Decoders

To read the speed of each wheel, the encoders TODO were used. Specs. To convert the signals from each encoder, hardware implemented decoders were written. The decoders read A and B pulses from each encoder and counted the time between each pulse to calculate the rotational speed of the wheels. This value was written on an AXI bus address for the OS to read. The hardware code was written in VHDL and can be seen in Appendix B.

5.4 RTOS tasks

5.4.1 Longitudinal control

longitudinal_control(), due to the limitations by the PWM, the control frequency was capped at 50 Hz, meaning a period of 20 ms.

5.4.2 Lateral control

lateral_control(), due to the limitations by the PWM, the control frequency was capped at 50 Hz, meaning a period of 20 ms.

5.4.3   Data aggregation

data_aggregation(), this task needs to be activated with as high frequency as possible. 1 ms is the shortest period allowed in FMP.

5.4.4   Communication

communication(), this task is bounded by the period of the larger communication loop, more than 20 ms, used 20 ms

5.5   Processor scheduling

220 =0+45 14 11 =0+63 26 204 214 264 268 2910 2124 21316

5.6   System overview

Chapter 6

Results

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

6.1   SafeG overhead

To measure the overhead of SafeG monitor, a hardware timer was started on the secure side, immediately followed by a syscall to switch OS. The non-secure side immediately read the value of the timer and printed this value. This procedure was looped many times to gather a reliable set of data points. Pseudo code for this procedure follows below:

Secure side:
   loop
      $start\_timer()$
      $safeg\_syscall\_switch()$
   end loop
Non-secure side:
   loop
      $time \leftarrow get\_time()$
      $reset\_timer()$
      $print(time)$
      $safeg\_syscall\_switch()$
   end loop
This resulted in a data set of 103211 execution times, see table 6.1.

Samples	103211
WCET	3.06 μs
BCET	0.96 μs
Average ET	1.63 μs
Median ET	1.72 μs
Standard deviation	0.258 μs

Table 6.1: Execution times of OS switch.

6.2 OS tasks

Communication, Lateral control, Data aggregation, Longitudinal control

6.3 Task switch

The task switch introduces extra 1.18 μs. Combined WCET: $4.24 \mu s$.

6.4 Hypervisor robustness

Unregistered exception was possible to create by accessing restricted AXI bus address, causing the processor to stop.

Fork bomb

Chapter 7

Discussion

Discussion about the results produced by the thesis.

7.1 Problem/Research

7.2 Information retrieval

Printing to SYSLOG takes time and has an effect on the system that is being monitored.

Timer used requires very little processor time, but still some. TOPPERS might have measured using less processor time.

7.3 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.

7.4 Errors

Unregistered exception, accessing restricted AXI bus address. Should be avoided by accessing via OS instead of address. All address space should be

handled by the OS/monitor, allowing or restricting access to certain address spaces. Probably not correctly setup.

7.5 Overhead resource loss vs resource gain

Higher frequency leads to more overhead. At what point is it not worth it to have a non-critical system application on the same hardware?

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] Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, February 2015.

[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] Data61. The sel4 microkernel, May 2017. `http://sel4.systems/Info/Docs/GD-NICTA-whitepaper.pdf`.

[12] Dictionary.com. safety-critical system, January 2017. `http://www.dictionary.com/browse/safety-critical-system`.

[13] Viktor Do. *SICS Thin Hypervisor Reference Manual Version 0.4*, April 2013.

[14] Software considerations in airborne systems and equipment certification. Standard, Radio Technical Commission for Aeronautics, Washington, DC, USA, December 2011.

[15] Encyclopedia.com. safety-critical system, January 2017. `http://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/safety-critical-system`.

[16] Google. Google trends. `https://trends.google.com/trends/`.

[17] Anne Håkansson. Portal of research methods and methodologies for research projects and degree projects, July 2013.

[18] Functional safety of electrical/electronic/programmable electronic safety-related systems – parts 1 to 7. Standard, International Electrotechnical Commission, Geneva, CH, April 2010.

[19] IEC/IEEE International Standard - Behavioural languages - Part 1-1: VHDL Language Reference Manual. Standard, IEEE, May 2011.

[20] IEC/IEEE Behavioural Languages - Part 4: Verilog Hardware Description Language (Adoption of IEEE Std 1364-2001). Standard, IEEE, 2004.

[21] 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.

[22] ITRON Committee, TRON Association. uITRON4.0 Specification Ver. 4.00.00. `http://www.ertl.jp/ITRON/SPEC/FILE/mitron-400e.pdf`.

[23] Leonard Kleinrock. Analysis of a time-shared processor. *Naval Research Logistics Quarterly, 11:1*, March 1964.

[24] Michael P. Lammert, Adam Duran, Jeremy Diez, Kevin Burton, and Alex Nicholson. Effect of platooning on fuel consumption of class 8 vehicles over a range of speeds, following distances, and mass. *SAE Int. J. Commer. Veh.*, 7:626–639, 09 2014.

[25] Max Lemke et al. Mixed criticality systems. report from the workshop on mixed criticality systems, February 2012.

[26] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM Volume 20 Issue 1*, pages 46–61, January 1973.

[27] Anna Lyons and Gernot Heiser. Mixed-criticality support in a high-assurance, general-purpose microkernel. 2014. `https://www-users.cs.york.ac.uk/~robdavis/wmc2014/2.pdf`.

[28] Daniel Potts, Rene Bourquin, Leslie Andresen, June Andronick, Gerwin Klein, and Gernot Heiser. Mathematically verified software kernels: Raising the bar for high assurance implementations, July 2014. `http://sel4.systems/`.

[29] SafeCOP. SafeCOP - part B, October 2016.

[30] Daniel Sangorrin Lopez. Advanced integration techniques for highly reliable dual-os embedded systems, July 2012.

[31] Daniel Sangorrín, Shinya Honda, and Hiroaki Takada. Dual operating system architecture for real-time embedded systems, Jul 2010. `http://www.artist-embedded.org/docs/Events/2010/OSPERT/slides/1-1_NU_OSPERT2010.pdf`.

[32] Scania. Platooning saves up to 12 percent fuel, December 2015.

[33] Sierraware. Sierravisor, May 2017. `https://www.sierraware.com/arm_hypervisor.html`.

[34] TOPPERS Project, Inc. Introduction to the TOPPERS/FMP kernel, February 2017. `https://www.toppers.jp/en/fmp-kernel.html`.

[35] TOPPERS Project, Inc. TOPPERS SafeG, February 2017. `https://www.toppers.jp/en/safeg.html`.

[36] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, 2007.

[37] W. Weber, A. Hoess, F. Oppenheimer, B. Koppenhöefer, 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.

[38] 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`.

[39] Wikipedia.com. Life-critical system, January 2017. `https://en.wikipedia.org/wiki/Life-critical_system`.

[40] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing System*.

[41] Xilinx Inc. Vivado Design Suite, March 2017. `https://www.xilinx.com/products/design-tools/vivado.html`.

[42] Xilinx Inc. Zynq-7000 All Programmable SoC, February 2017. `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`.

[43] 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.