# Platform Independent Safety-Critical Operating System

Emil Delic, Karolin Löser, Ali Hayek, Josef Börcsök
Computer Architecture and System Programming
Wilhelmshöher Allee 71 - 73
D-34121 Kassel, Germany
{emil.delic, karolin.loeser, ali.hayek, geloglu}@uni-kassel.de

*Abstract*— **An end-element, a logic solver system and a sensor are essential parts of safety-critical systems. The time to market of a brand new logic solver system might drastically increase when a safety-critical operating system needs to be written from scratch. According to our observations, the commercial safety-critical operating systems are not flexible enough to be reused for a brand-new platform. Consequently, we are going to present a generic safety-critical operating system. The operating system is written in an object-oriented programming language. Inheritance and polymorphism are used to achieve hardware independency. The first results show that more than thirty percent of source code might be reused. Thus, a lot of time for testing, documenting and certifying can be saved.**

***Keywords—safety-critical software; real-time operating system; IEC-61508; System-On-Chip; certification***

## I. INTRODUCTION

Safety-critical systems have a pivotal role in railway, automobile, oil and gas industry. The term safety-critical system refers to a system which failure yields to human casualties and/or loss of property. Moreover, safety-critical software is a fundamental property of a safety-critical system. In recent years, there has been increasing interest in safety-critical software development. For example, new cars have more electronic components compared to older cars. In the light of recent events in safety-critical software development, it is becoming extremely difficult to have end product finished in time. Surveys, such as the one conducted by UBM Tech (2014), showed that more than 50% of projects are finished late or are cancelled [1]. Furthermore, Pumfrey (1999) reports that software development costs are significantly bigger as the cost of hardware development [2].

One of the greatest challenges is to certify a product. The term product here refers to a hardware platform with software implementation. Certification is always in accordance to a standard. One widely applicable standard for electrical/electronic/programmable electronic safety-related systems is the IEC-61508 Standard [3]. Instead of defining dozens of safety parameters, the Standard [3] defines a SIL term. The SIL stands for the safety integrity level. There are four levels from SIL1 up to SIL4. The SIL1 is the lowest and SIL4 the highest level. Furthermore, the Standard [3] identifies the techniques and measurements that need to be implemented to achieve desired SIL level.

This paper describes the design of a generic operating system in accordance to IEC61508. The specific objective of this design is to save time necessary for documenting, testing and certifying parts of the operating system. Part of the aim of this paper is to define core parts of the operating system, which have to be independent from the underlying hardware architecture. The reader should bear in mind that the proposed operating system is not supposed to replace other available and commercial solutions. Rather, this research focuses on brand-new hardware architectures, where software support is not available or very constricted. As a result, an operating system for these architectures must be implemented from scratch. Therefore, the proposed design might be utilised to speed up time to the market.

## II. PROBLEM STATEMENT

In recent years, there has been an increasing interest in the safety-critical hardware development and there is a strong possibility that this trend will continue in the future. For example, a company is developing a safety-critical hardware. If the hardware does not have satisfactory software support, the company will likely end up with developing a safety-critical operating system. As a result, the time to market will increase. The main challenge faced by the operating system developers is the fact that they will start from scratch. According to our research, none of the available safety-critical operating systems can be reused for a brand new hardware platform. Our research of the commercial safety-critical operating systems is covered in the next chapter. This study provides some important insights into possibilities decreasing the time to the market by introducing a generic safety-critical operating system core. The operating system core is going to be certified up to SIL3 and it is not a standalone operating system. Rather, it is a kind of help for an operating system integrator. The operating system integrator can concentrate on integrating hardware-dependent parts. The hardware-independent parts are already given in the generic operating system core. Moreover, these parts shall be tested, certified and well-documented. After integration, the application developer can use the operating system, which might offer a layer of abstraction and allows focusing on the application.


978-1-4673-7186-5/15/$31.00 ©2015 IEEE

## III. COMMERCIAL SAFETY-CRITICAL OPERATING SYSTEMS

A considerable amount of commercially available safety-critical operating systems has been examined. Table 1 provides an overview of the evaluated safety-critical operating systems. Some of them are certified up to SIL3. However, our observations have indicated that many of them are meant to be used for the single-core architecture. As shown in Table 1, many of them are only certified for a defined number of platforms.

TABLE I. SAFETY-RELATED OPERATING SYSTEMS

| Name | Certification | Remark | Reference |
|---|---|---|---|
| PikeOS | no | supports: PowerPC, MIPS, IA-32 does not support: multiprocessor architectures | [4] |
| Sciopta | up to SIL3 | supports: ARM, PowerPC, Coldfire does not support: multiprocessor architectures | [5,6] |
| SafeRTOS | up to SIL3 | supports: 32 bit microcontrollers | [7] |
| VXWorks | up to SIL3 | supports: PowerPC, Intel, ARM | [8,9,10] |
| Deos OS | DO-178 Level A | supports: x86, PowerPC, ARM, MIPS | [11] |
| Lynx OS | DO-178 Level B | ARM, PowerPC, x86, MIPS | [12] |
| uC/OS-II | up to SIL3 DO-178 Level A | supports: some of ARM, Cortex, PPC architectures | [13] |

## IV. GENERIC SIL3 SAFETY-CRITICAL OPERATING SYSTEM

The main purpose of this paper is to present a concept of an operating system core. The operating system core is going to be certified up to SIL3 and is going to be used in safety-critical environment. In addition, it is hardware-independent. The SIL3 environment consists of an operating system, the hardware, the hardware drivers and an application. Obviously, the application and the hardware drivers are not implemented by the generic operating system core. However, the operating system uses the hardware drivers to communicate with the underlying hardware. Furthermore, the application is implemented by an application developer. In contrast to the integrator, the application developer doesn't implement low-level interfaces. In conclusion, the operating system has open-ends, the hardware drivers from one side of the operating system and the application from another side, which need to be linked.

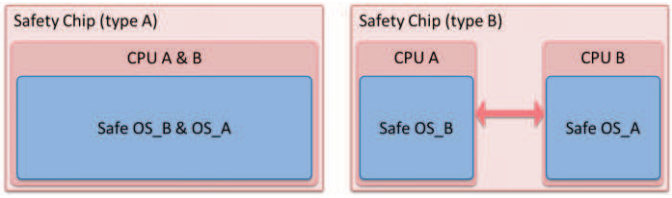


Figure 1. Two ways to implement 1oo2 architecture

In order to understand the implementation of the operating system, the architecture of the SIL3 hardware needs to be defined. A recent study by Hayek and Börcsök [14] examined safety chips with on-chip redundancy. The findings of this study suggest that most of the safety chips are using the 1oo2 architecture for achieving SIL3 level. Figure 1 provides two possible ways to implement 1oo2 architecture. On way is to implement a hardware comparator, "type A" on figure. As a result, the operating system integrator can't program both CPUs. The operating system is identic for both CPUs. The second way is to use a software comparator, "type B" on figure. The CPUs are independent and the integrator must program both CPUs.

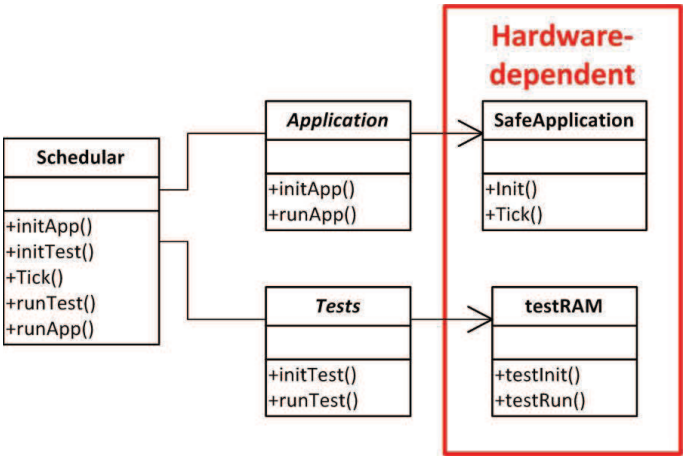


Figure 2. The primiary class diagram of the operating system

As can be seen from figure 2, the operating system core consists of a Scheduler and the generic interfaces. The scheduler calls the hardware tests and the application. The calls are virtual functions and they are part of the generic interfaces. This can be illustrated by the implementation of a test class. The test class has a method runTest. The runTest method is a virtual method and it doesn't have an implementation; consequently, the test class is an abstract class. The child classes of the test class are going to be implemented by the integrator. The tests are hardware-dependent. Hence, they are not part of the operating system. For example, the processors' registers need to be tested using some patterns, like 0xAA and 0x55 or walking bit. Such a test is used for detecting bit failures. This test is defined in the IEC-61508 Standard as well as other techniques and measures to detect random hardware failures.

Next, the application class is also a derived class of an abstract class. The abstract class has two virtual methods: a method for the initialization and one for the cyclic calls. The

application developer is going to derive an application class from the abstract class.

Further, the IEC-61508 Standard describes temporal and logical sequence monitoring. The aim is to find error-prone program sequences. Sequences can be called at the wrong time or in the wrong order. The operating system monitors the correct execution sequence of all modules, e.g. counting checkpoints to allow detecting an incorrect jump inside of a function. Also, the operating system performs logical and temporal monitoring.

The IEC-61508 Standard suggests that the software shall be modular, testable and capable for safe modification. Because of this, the operating system is written in an object-oriented programming language. Henceforth, useful design patterns will be identified and used in the operating system. Gamma et al. (1994) set up a catalog of design patterns [15]. Thus, using proper design patterns and object-oriented techniques, such as use of classes and inheritance, we might achieve high modularity, testability and capability for safe modification. Moreover, the classes of the operating system are grouped to functional groups. For example, if one target system does not have an Ethernet interface, an Ethernet class will not be linked and downloaded to the target. This is very important for targets with a small memory. In this way, independency is not only guaranteed vertically, but also horizontally.

## V. First Results

In order to assess the operating system, two hardware platforms have been used. The first platform is the HiCore1 chip from HIMA [16]. It is a SoC (system on chip) safety system and consists of three DP80390 processors. Two of them are forming the 1oo2D architecture. The third processor is working as a communication processor. Each processor has its own memory. The connection between safe and non-safe part is without interference. Moreover, HiCore1 is certified by TÜV Rheinland up to SIL3 [16]. HiCore1 has a hardware comparator, two DP80390 processors are coupled. Because of that, the HiCore 1 is going to be utilised to integrate type A of the operating system.

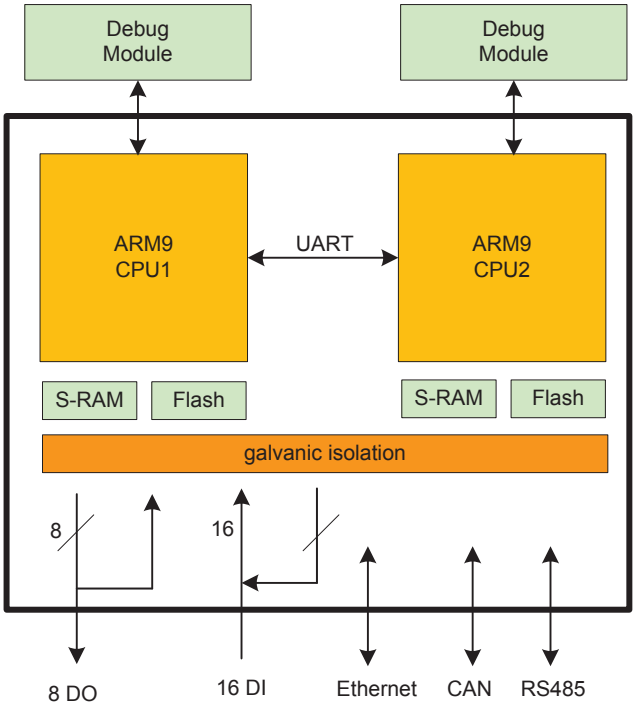


Figure 3. The ARM9 System

Figure 3 shows the second hardware platform, an ARM9 system. This system does not contain a hardware comparator. Therefore, on the ARM9 system type B of the operating system is going to be installed. The software comparator is implemented by the operating system. As shown in Figure 3, the ARM9 system has two debug modules. Thus, the operating system needs to be installed on both of them separately. The operating system consists of more modules: scheduler, application and tests. They are core parts of the operating system. Whereas, the hardware-dependent modules are: user_application and tests. Other modules may involve the low level initialization.

In order to assess the operating system, the size of the source codes has been measured. To compare the scores, the number of used lines has been used. Over thirty percent of the source code belongs to the hardware independent modules. The rest of it belongs to the test modules, e.g. hardware dependent modules. It can be shown that the core parts of the operating system are equal for both hardware platforms.

In summary, the core parts could be used for other platforms without changing them. As a result, the integrator is going to spare time for documenting, testing and certificating the core parts. However, the integrator will have to implement the tests to detect random hardware failures using the techniques and measurements described in the IEC-61508 Standard [3] and certify it according to the standard. Finally, the application developer will have to implement, test and certify a safe application.


978-1-4673-7186-5/15/$31.00 ©2015 IEEE

## VI. Future work

To determine whether the use of the operating system core reduces the time to market, other possibilities will be evaluated. For example, we might to try a single program instead of the operating system. The program might be written from scratch. Secondly, at least two different hardware architectures are going to be evaluated. Two questions will have to be answered:

1. Is it possible to use any of the available operating systems for a brand-new hardware platform? If yes, how much time it will take to certify it?
2. Does the use of the generic certificated safety-critical operating system core decrease the time to the market?

## VII. Conclusion

The main goal of the current study was to present the design of a safety-critical operating system. This study has identified the core parts of the operating system: the scheduler and the generic interface. More than thirty percent of the source code belong to the scheduler and the generic interfaces. Those parts are hardware independent. An implication of this is the possibility that core parts of the operating system can be reused. As a result, the time for testing, documenting and certifying can be saved. Whilst this study did not confirm that the operating system is better than other commercial solutions, it did partially substantiate that the core parts can be compiled with miscellaneous compiler and can be downloaded to the several hardware platforms without changes. These results suggest that the operating system can be reused and more than thirty percent of the source code does not need to be rewritten.

## References


[1] “2014 Embedded Market Study: Then, Now: What’s Next?”, UBM Tech, 2014.

[2] D.J. Pumfrey, “The principled design of computer system safety analyses”, University of York, 1999.

[3] International Electrotechnical Commission, IEC/EN 61508: International standard 61508 functional safety: safety related systems: Second Edition, Geneva, 2010.

[4] R. Kaiser and S. Wagner, The PikeOS Concept: History and Design, SysGO AG White Paper, 2007.

[5] “SCIOPTA - Real-Time Kernel”, SCIOPTA Systems AG.

[6] “SCIOPTA - Real-Time Kernel - Reference Manual”, SCIOPTA Systems AG.

[7] “SafeRTOS User Manual for the code composer studio TMS570 MPU product variant”, WITTENSTEIN HighIntegritySystems, 2011.

[8] “Wind River VXWORKS 653 Platform 2.4 and 2.5”, Wind River Systems, 2015.

[9] “Safety Profile for VXWORKS”, Wind River, 2015.

[10] “Wind River VxWorks Cert Platform”, Wind River.

[11] “DeOS A Time & Space Partitioned DO-178 Level A Certifiable RTOS” DDC-I Safety Critical Software Solutions for Mission Critical Systems.

[12] Lynx Software Technologies, "LynxOS-178 RTCA/DO-178B, level A Certified RTOS and FAA-accepted Reusable Software Component", Lynx Software Technologies, Inc., 2014.

[13] Micrium.com, 'Why Certification? | Micrium', 2015. [Online]. Available:http://micrium.com/certification/why-safety-critical-certification/. [Accessed: 04- May- 2015].

[14] Hayek, A.; Borcsok, J., "Safety chips in light of the standard IEC 61508: Survey and analysis," Fundamentals of Electrical Engineering (ISFEE), 2014 International Symposium on , vol., no., pp.1,6, 28-29 Nov. 2014, doi: 10.1109/ISFEE.2014.7050579.

[15] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software, " USA, Addison-Wesley, 1994.

[16] “HIMA ProductNews SIL 3 safety system-on-chip”, HIMA Paul Hildebrandt GmbH + Co KG, 2013.