Self Diagnostics and Isolation Mechanisms for Mixed Criticality Systems

Asmaa Tellabi, Christoph Ruland, Karl Waedt, and Sabri Abdelbast

Original scientific paper

***Abstract*—Virtualization is a technology that is frequently employed in computers and servers to provide isolation for execution environments, and to support the execution of multiple Operating Systems (OS) on the same hardware platform. In the embedded systems´ world, virtualization has been a rising trend, essentially because it offers an isolation mechanism that provides hardware manufacturer´ independence and it avoids obsolescence issues. The isolation mechanism supports safety and security measures, and assists in the certification of safety-critical systems. Virtualization offers improved performances, better transparency, portability and interoperability by integrating hardware and software resources, and also networking services into one computing entity. It makes the integration process of Mixed Criticality Systems (MCS) easier. For industries, Field-Programmable Gate Arrays (FPGAs) hardware solutions provide the needed level of flexibility and performance. In this paper, a Self-test application is integrated in the hardware and also in the software level. The importance of self-test applications for Instrumentation and Control (I&C) systems will be discussed in the context of virtualization. For this implementation a type 1 hypervisor called Xtratum is used. An analysis of inter-partition communication channels´ performance will be provided including the implications multicore approaches will have on communication. The novelty of this work is to study the isolation impact multicore approaches can have on inter-partitions communications in Xtratum. Another novel aspect is the implementation of a self-test application in the hypervisor and the board as well.**

***Index Terms*— Virtualization, MCS, Xtratum, Self-test, Cortex A9, FPGA, multi-core.**

I. INTRODUCTION

With the advance of digital technology, a remarkable increased worldwide attention towards FPGAs is seen, including safety and operational I&C applications in Nuclear Power Plants (NPPs) [1]. FPGAs are capable of increasing the reliability and also decreasing the complexity for some gate circuit logic-based hardware functionalities [1].

In [2], a new secure system´s architecture was presented based on Xtratum, which is used to implement the test architecture in this paper, it also presented the current use cases of virtualization technologies in the industry and the possible future implementations. Presently, MCS implementations are gaining popularity both in research and in the industrial design domains. As the computational demands are increasing, this offers multiple benefits by making the implementation of applications with different criticality levels into one system possible [2]. Nowadays, the embedded market is gradually willing to exploit the financial benefits that virtualization technology offers. Embedded systems are not similar to traditional computers. They have a fixed purpose and are created precisely to achieve a particular task. Currently, many of the developments on virtualization are dedicated to desktop systems. Consequently, using these results on embedded systems is not a simple task. As virtualization technologies are actively expanding, with various competing technologies gradually becoming mature, the perfect solution is not defined yet. Security levels that are essential to embedded devices differ based on the device´s function [3]. MCS rely on temporal and spatial partitioning [3].

Currently, various digital platforms integrate embedded diagnostics applications that are executed repeatedly during the platform´s operating phases. Self-diagnostics applications are capable of testing multiple modules that are available in the platform [4]. In [5], the different authentication and access controls mechanisms were presented, including security concerns surrounding industrial systems. As stated in [5], for industrial computers, Availability is the primary security concern and self-diagnostics applications can be used to verify the internal status. The idea of developing self-diagnostics applications are based on the findings in [5]. This type of application is also able of guaranteeing the platform´s safety integrity. In this paper, both applications developed on the hardware and software level focus on verifying the availability of components. The main focus will be on techniques to ensure availability characteristics. In [9], the architecture in [2] was further detailed with communication channels and communication path between partitions. A comparison between single core approaches and multicore was presented

Manuscript June 7, 2019; revised October 23, 2019. Date of publication November 18, 2019. Date of current version November 18, 2019. Some of the addressed cybersecurity related topics are being elaborated as part of Framatome GmbH's participation in the "SMARTEST" R&D (2015-2018) with German University partners, partially funded by German Ministry BMWi.

A. Tellabi is with the Department of Data Communications Systems at University of Siegen and with Framatome GmbH, Germany. C. Ruland is with the Department of Data Communications Systems at University of Siegen, Germany. K. Waedt is with the Department of Cybersecurity for I&C Systems at Framatome GmbH, Germany. A. Sabri is with the Department of Hardware Co-design at the University of Erlangen-Nürnberg, Germany (e-mails: asmaa.tellabi@student.uni-siegen.de, christoph.ruland@uni-siegen.de, karl.waedt@framatome.com, abdelbast.sabri@fau.de).

Digital Object Identifier (DOI): 10.24138/jcomss.v15i4.810

in that paper but only on the hardware level. This paper implemented the test architecture based on the information provided in that paper [9].

The remainder of this paper is structured as follows. Section II explains the different techniques used to implement MCS. Section III emphasizes the utility and implementation of Self-diagnostics applications on the hardware level. In section IV, experimental results on the inter-partition communication channels, using single-core and multi-core approaches, including the self-test application on the software layer are analyzed. Section V concludes the paper with an outlook of the future work.

II. MCS Techniques

A. Separation Kernel

Separation Kernel was first introduced in 1981 by John Rushby [6], in order to create a secure environment by providing temporal and spatial isolation for applications. An additional intent was to guarantee a single access point to the system by verifying that there are no other unauthorized channels for information flows between partitions that were implemented other than those created. The separation kernel is characterized by its small size; this will allow formal verification of its accuracy. The approach used for a separation kernel is based on Multiple Independent Levels of Security/Safety (MILS) [7]. MILS is a high-assurance security architecture based on isolation [6] and a controlled information flow. Fig.1 shows an architecture based on the MILS concept. MILS offers ways to have multiple strongly isolated partitions with different security/safety levels on the same physical entity. A separation kernel is considered as a method that provides security at the OS kernel level [6] by simulating a distributed environment. It is seen as a solution which creates and verifies large and complex kernels that are intended to offer multilevel secure operations on systems. Based on separation kernels, the system´s security is realized partially by physically isolating different modules and managing trusted functions which are executed within some modules. Separation kernel is expected to provide hosted applications with high-assurance partitioning and controlled information flows that are tamperproof and also non by-passable.

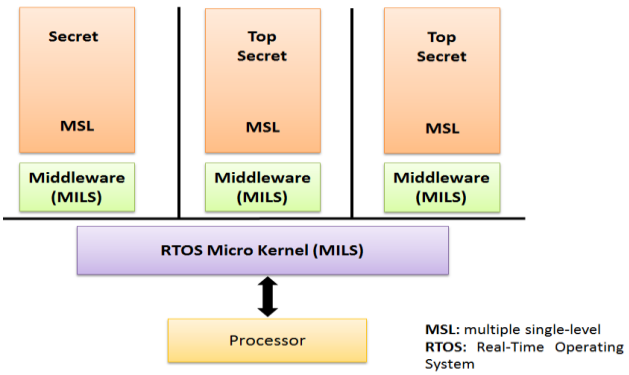

Fig. 1. MILS concept.

Hardware Virtualization is an approach that can solve the issue of resources´ sharing. In such configurations, the hypervisor is considered as a secure and trusted element that manages all of the system´s resources [8]. The hypervisor is placed at the highest privilege level in the system and controls the access to system´s resources by virtual machines (VMs). A hypervisor can be seen as a container that comprises each VM in a separated environment and does not permit the propagation of errors, by this decreasing the surface of attack. Conversely, this signifies that the system´s security is as robust as the hypervisor itself. For this reason it is recommended to reduce the hypervisor´s size to simplify the process of verification and validation. Hypervisors can be divided into two categories [2]:

1) *Type I Hypervisors*

Native or bare-metal hypervisors, they are executed directly on top of the hardware to control the hardware and to manage the guest OS. They are executed in a privileged processor mode, e.g. ring 0 or supervisor mode [8]. Therefore, supplementary overhead and security matters, which could be caused by an underlying OS, are not present. Instead, the Hypervisor manages the entire hardware, e.g., scheduling and resource´s allocation, so it might get complex. Since it runs directly on the machine´s hardware, it is seen as an OS or a kernel that integrates features to support VMs [2]. Fig2. shows a Type 1 hypervisor´s architecture. This means that a Type-1 hypervisor can be much larger than a Type-2 hypervisor, because of the additional code that is needed to integrate these features.

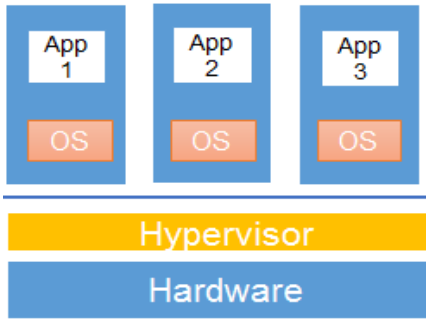

Fig. 2. Type 1 or Bare-metal Hypervisor Architecture.

2) *Type II Hypervisor*

It is executed on top of an OS, so it can use all services that are available in the OS. One obvious drawback is the considerable overhead caused by the underlying OS and any present issue in that OS which can directly affect the VM [9]. A Type-2 hypervisor should be simpler than a Type-1 hypervisor because the memory management, scheduling task , resource allocation, and hardware drivers of the used host OS are already implemented in it. Figure 3 shows the differences between these hypervisors´ architectures. A Type-2 hypervisor offers only virtualization support services. This type virtualizes the real machine even if the hypervisor is being executed as an application in the host OS [7]. The most efficient type of hypervisors to deploy depends on the use cases. Bare metal hypervisors offer higher performance, availability, and security than Type-2 hypervisors, because they do not rely on an OS layer [6]. One disadvantage, which is a Type-2 advantage, is that Type-1 hypervisor hardware support is restricted to exactly what the hypervisor was created for and to certain boards only, while Type-2 hypervisors utilize OS drivers to abstract any hardware [7]. Ease of use is also a problem, since Type-2 hypervisors are running over a familiar OS user interface.

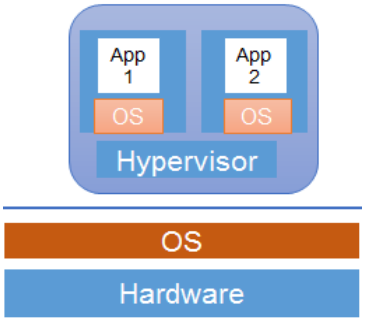

Fig. 3. Type 2 or Hosted Hypervisors Architecture.

III. Self Diagnostics Applications

B. *Self-diagnostic Application for I&C Systems*

A built-in self-test (BIST) or a self-diagnostic application is a procedure, which allows a system to test itself. They were created to fulfill stringent requirements such as high availability, low latency in the operation cycle and low cost testing procedures during operation [10]. Multiple digital platforms integrate diagnostics applications that run on a continuous basis in the platform´s operating cycles [4]. These applications are able to test multiple system components as well as application parameters. They can be used to examine the internal calibration and define the health status. These diagnostics´ functions help with checking the platform´s integrity. Computer systems based on a fail-safe design that are vital to safety will certainly integrate on-board self-diagnostics applications to guarantee the timely detection of failures and errors. As indicated before, these on-board self-diagnostics applications are generally created to check the memory integrity, input/output abilities, processor´s availability, etc [4]. Diagnostic´s procedures are generally divided between the initialization phase that occurs before the start of control systems (called self-test during startup) and applications, and the diagnostic´s phase that is executed in parallel with the application code (called cyclic selftest) [1].

A self-diagnostic application would deliver a comprehensive analysis of the failure process inside boards. In the United Kingdom, on-board self-diagnostics are integrated inside protection systems for the advanced gas cooled reactor (AGC) fuel roads and the primary protection system (PPS) for Sizewell B NPPs [1]. The industry must gradually exploit these benefits from digital systems. Regulators still have a conservative position on this matter since only few applications have been integrated; consequently a deep understanding and confidence in digital abilities of computer based systems are not yet completely achieved.

C. *Xtratum*

Xtratum is a bare-metal or type 1 hypervisor, that was designed to realize temporal and spatial separation for safety critical applications. It was created by the Universidad Politécnica de Valencia in Spain with contributions from the Lanzhou University in China [11]. Xtratum offers virtualization services to partitions. It runs in supervisor processor mode and virtualizes the Central Processing Unit (CPU), memory, interrupts, and other available peripherals. It was designed precisely to fulfill safety critical demands. Iinitially, it was created to run on an x86 processor architecture (version 2.0), then it has been fully altered to run on a SPARC version 8 architecture, to be precise for the LEON2 and LEON3 processors. Currently, a new version has been uploaded that is compatible with Advanced RISC Machine (ARM) processors. The present version consists of the features needed to develop safety critical systems built on ARINC 653, AUTOSTAR and other standards [2]. Used for the aerospace sector, Xtratum is capable of creating software blocks on board, and managing payloads units in aerospace. Xtratum implements an ARINC 653 scheduling policy, it provides partition management, inter-partition communication, health monitoring and traces, so it adheres to the ARINC standard´s specification. The hypervisor is configured using an eXtensible Markup Language (XML) configuration file, where system´s resources are allocated statically [9][12].

D. *Implementation at the Hardware and Software Level*

A self-test application was implemented on the board based on the tutorial in [10], which was modified in order to be able to run on a Cortex A9 processor and also to be able to run tests every minute. Fig 4 shows the execution of the Universal Asynchronous Receiver/Transmitter (UART) test. Multiple tests are performed on the Processing System (PS) or cortex A9:

1) PS UART Test

This test case executes a local loopback and checks that data can be sent and received.

2) PS I2C Test

This test case is dedicated to the Inter-Integrated Circuit (I2C) component. In case the test was successful, the device will be reset.

3) PS Timer Test

This test frees the timer enabled bit in the control register, writes in the timer load register and verifies if the read value equals the value written. Then, it restores the control register along with the timer load register.

4) PS Snoop Control Unit/Generic Interrupt Controller (SCU/GIC) Test

This test reads the ID registers and then compares them.

5) PS Device Configuration Interface (DCFG) Test

It does a self-test on the Device Configuration device and the xdevcfg driver. The purpose is to demonstrate the use of the xdevcfg driver.

6) PS Double Data Rate Type3 (DDR3) Test

It tests memory regions, memory controller, sizes and regions that are specified.

7) PS Interrupt Test

It checks if the SCU Private WDT driver and hardware in Timer mode are still functioning through interrupts.

8) PS Watchdog Timer Test

It does a basic test on the watchdog timer device and driver. The purpose of this function is to show how the watchdog Timer driver can be used.

9) PS Light Emitting Diode (LED) Test

It uses an application that tests if LEDs blinks on the board.

10) PS General Purpose Input/Output (GPIO) Switch Test

It includes an example for using GPIO hardware and driver. For this example, it considers that a UART device or standard input output (STDIO) device is present in the hardware system.

```
**********************************************************
**********************************************************
**    Xilinx Zynq-7000 AP SoC ZC706 Evaluation Kit      **
**********************************************************
**********************************************************
Wait 1 min
New Test
Choose Feature to Test:
1: PS UART Test
2: PS IIC Test
3: PS TIMER Test
4: PS SCUGIC Test
5: PS DCFG Test
6: PS DDR3 Memory Test
7: PS Interrupt Test
8: PS Watchdog Timer Test
9: PL LED Test
A: PL DIP SWITCH Test
B: PL Push Button
0: Exit
choice: 1

**********************************************************
**********************************************************
**                  ZC706 - UART Test                   **
**********************************************************
**********************************************************
Testing UART
115200,8,N,1
Hello world!
UART Test Passed

choice: 2
```

Fig. 4. The execution of the UART Test.

In order to implement the self-test application on the software layer, the compatible version of Xtratum with ARM processors was installed [11] running on Ubuntu 15.4. This implementation was based on the architecture shown in Fig. 5. A self-test application on Xtratum is capable of detecting if applications inside partitions are still responding or not, meaning that it checks the partition´s internal status. To implement this self-test application on Xtratum, the Xtratum Abstraction Layer (XAL) API's *XM get partition status ()* was used, it returns the current state of the system. In case applications inside partitions are not responding, it shuts down the partition and restarts it again. A partition is considered not responding when the function *XM get partition status ()* returns a negative value.

First, the XML file had to be configured. System partitions and user partitions had to be specified; also the path for communication channels and the scheduling plan of these partitions had to be included in the XML file. In this case, the schedule plan provided in Table 1 was followed. Xtratum defines two types of partitions: User and System [2]. Fig.6 shows the configuration of the XML file for the single core approach. System partitions are permitted to manage and control the state of the system along with other partitions. System partition´s rights are related to the ability of managing the system, but not the ability to access directly the native hardware neither to break the isolation. A system partition is scheduled in the same way as a user partition; and it is allowed to only use the resources allocated to it in the XML configuration file. In any system, applications communicate between them and to the outside world as well. In this example, only inter-partition communication mechanisms were considered. Xtratum offers two types of inter-communication channels [2][9][13]:

1) Queuing Channel

It is deployed for buffered unicast inter-communication. As a result, each port has a queue to store messages until they are delivered to the destination partition. The delivery of these messages is ensured by a First in First out (FIFO) mechanism.

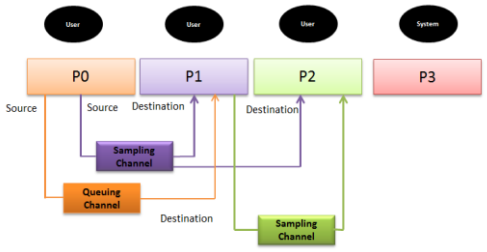

Fig. 5. System´s architecture.

TABLE I.
SCHEDULE PLAN

Partition	Type	Duration
P0	User	1 ms
P1	User	1 ms
P2	User	1 ms
P3	System	1 ms

2) Sampling Channel

It is implemented for broadcast, multicast or unicast inter-communications. This channel does own a queue. Consequently, messages stay in the source port until they are read or overwritten by a new instance. This feature is mandatory in case the destination partition needs information about the current situation.

```
<SystemDescription name="channels" version="1.0.0" xmlns="http://www.xtratum.org/xm-arm-2.x">
 - <HwDescription>
   - <MemoryLayout>
       <Region size="1024MB" start="0x00000000" type="sdram"/>
     </MemoryLayout>
   - <ProcessorTable>
     - <Processor frequency="400Mhz" id="0">
       - <CyclicPlanTable>
         - <Plan id="0" majorFrame="4ms">
             <Slot start="0ms" id="0" partitionId="0" duration="1ms"/>
             <Slot start="1ms" id="1" partitionId="1" duration="1ms"/>
             <Slot start="2ms" id="2" partitionId="2" duration="1ms"/>
             <Slot start="3ms" id="3" partitionId="3" duration="1ms"/>
           </Plan>
         </CyclicPlanTable>
       </Processor>
     </ProcessorTable>
   - <Devices>
       <Uart name="Uart" id="1" baudRate="115200"/>
     </Devices>
 </HwDescription>
```

Fig. 6. XML Configuration file for the single-core approach.

```
void PartitionMain(void) {

    int retValue;
 retValue = XM_reset_partition(P0, -1, XM_WARM_RESET, 0);
    PRINT("Restarts P%d (return value %d)\n", P0, retValue);
    if (retValue >= 0)
        PrintStatus();
    XM_idle_self();

    retValue = XM_reset_partition(P1, -1, XM_WARM_RESET, 0);
    PRINT("Restarts P%d (return value %d)\n", P1, retValue);
    if (retValue >= 0)
        PrintStatus();
    XM_idle_self();
    retValue = XM_reset_partition(P2, -1, XM_WARM_RESET, 0);
    PRINT("Restarts P%d (return value %d)\n", P2, retValue);
    if (retValue >= 0)
        PrintStatus();

    XM_idle_self();

    PRINT("Halting System ...\n");
    retValue = XM_halt_system();
}
```

Fig. 7. Self-test code in Xtratum.

To implement this test architecture, a Sampling Channel between partitions P0 (source), P1 (destination), P2 (destination) and P3 which is the self-test application were created. For Queuing Channels, one channel between P0 (source) and P1 (destination) was implemented. For this implementation, P0 owns a write access to Port Q (Queuing Port) and Port S (Sampling Port). P1 owns only a read access to Port Q (Queuing Port), to Port S (Sampling Port) and has access to the Shared Memory. P2 has a read access to Port S (Sampling Port) and to the Shared Memory. Fig.7 shows a part of the self-test code. The compilation of partitions is done using the command "*make*". The result of this command includes an image of the resident software (RSW) that has to be put on top of the board and the container [13].

E. Experimental Results

11) Single-core Approach

XAL API's XM_get_time() is a method provided by Xtratum that measures time performance. It returns the hypervisor's internal incremental clock in microseconds (us) as a 64 bit unsigned integer. It was declared before and after a message´s transfer. The difference was calculated and used as the transfer's delay. In this implementation only bare ANSI C applications were implemented. 10 messages were sent using methods provided by Xtratum for inter-partitions communication. The message size is 128 bytes. For Queuing messages, the Queue depth was 16 messages deep.

TABLE II
TOTAL DELAY ON XAL SAMPLING AND QUEUING PORTS USING A SINGLE-CORE APPROACH

Number of messages	Sampling Messages Delay (us)	Queuing Messages Delay (us)
0	4028	6793
1	4028	10142
2	4028	11540
3	4028	6783
4	4028	6793
5	4028	11540
6	4028	11525
7	4028	11515
8	4028	7975
9	4028	6786
Minimum	4028	6783
Maximum	4028	11540
Average	4028	9139,2

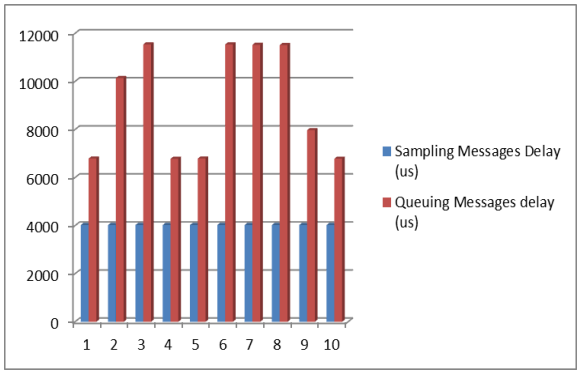

Fig. 8. Delays on Sampling and Queuing ports using the single-core approach.

Table 2 presents sampling and queuing message delays of each message. This implementation was made on a Xilinx ZC 706 board. As Table 2 and figure 8 demonstrate, Sampling Messages are faster than Queuing Messages; this delay is caused by the internal structure of ports and the procedure used for sending messages. For Queuing Messages, each port is connected to a queue where messages are stored until they are delivered to the destination partition. Transmission of messages is done in a First-in First-out (FIFO) order [13]. This is the reason behind delays in Queuing Messages, because it must access a FIFO rather than a direct memory place. For Sampling Messages, they stay in the source port until they are transferred over the channel or are overwritten by a new message. The recipient then enters the specified memory, configured in an XML File, and reads messages.

12) Multi-core Approach

Generally, multicore processors comprise two or more cores that work in parallel in order to read and execute instructions. These computational units or cores are integrated on a single processor. One obvious advantage of multicore processors is that they offer similar or better performance as compared to a single processor but with lower power consumption and at a reduced clock frequency by executing more instructions in parallel [14]. This processing rapidity is the result of the internal composition in processors; a multi-core processor consists of numerous cores that execute simultaneously instructions, with a lower frequency than a processor featuring a single core would process [9]. Using the same clock frequency, a multicore processor will execute more tasks than the single core processor would do. Furthermore, multicore processors offer better performance and execute more instructions, thus consuming less power as compared to single core processors; this can be a fundamental aspect for devices like mobile phones or laptops that work with batteries [15]. In this section, a multicore configuration was used in Xtratum to analyze the implications it will have on inter-partitions communication performance.

For Xtratum´s configuration, two approaches can be used, either a single core approach or a dual-core approach. With the single core approach, only one virtual core was used in Xtratum [16]. With the multi-core approach in Xtratum, only 2 virtual cores can be used. Virtual cores´ configuration is done internally, in the XML configuration file as shown in Fig.9. For this implementation, [P0, P1] are integrated in core1 and [P2, P3] are integrated in another separated core. Fig.10 shows the selected architecture for this dual core implementation. Xtratum has been used in multiple EU projects such as DREAMS [17] and OVERSEE [18].

In this implementation, the previous scheduling policy shown in Table 1 related to the single core implementation was used. As shown in Table 3, the total delay of Sampling and Queuing channels was reduced. The delay on sampling channels was reduced by 0.03%. On the other hand, the delay on queuing channels was reduced by 24.6%. In the single core architecture, Xtratum is responsible of virtualizing physical CPUs in the hardware to partitions. A partition uses the virtual CPU (vCPU) to execute the code inside the partition.

```
<?xml version="1.0"?>
<SystemDescription name="channels" version="1.0.0" xmlns="http://www.xtratum.org/xm-arm-2.x">
 - <HwDescription>
    - <MemoryLayout>
        <Region size="1024MB" start="0x00000000" type="sdram"/>
      </MemoryLayout>
    - <ProcessorTable>
       - <Processor frequency="400Mhz" id="0">
          - <CyclicPlanTable>
             - <Plan id="0" majorFrame="4ms">
                  <Slot start="0ms" id="0" vCpuId="0" partitionId="0" duration="1ms"/>
                  <Slot start="1ms" id="1" vCpuId="0" partitionId="1" duration="1ms"/>
                  <Slot start="2ms" id="2" vCpuId="1" partitionId="2" duration="1ms"/>
                  <Slot start="3ms" id="3" vCpuId="1" partitionId="3" duration="1ms"/>
               </Plan>
            </CyclicPlanTable>
         </Processor>
      </ProcessorTable>
    - <Devices>
        <Uart name="Uart" id="1" baudRate="115200"/>
      </Devices>
  </HwDescription>
```

Fig. 9 XML Configuration file for the multi-core approach.

TABLE III
TOTAL DELAY ON XAL SAMPLING AND QUEUING PORTS USING A MULTI-CORE APPROACH

Number of messages	Sampling Messages Delay (us)	Queuing Messages Delay (us)
0	4026	6889
1	4027	6888
2	4028	6887
3	4026	6889
4	4027	6888
5	4028	6896
6	4028	6889
7	4027	6896
8	4026	6889
9	4026	6897
Minimum	4026	6886
Maximum	4028	6897
Average	4026,9	6890,8

Then, the hypervisor initializes the CPU, and after the initialization phase the scheduling plan will be started. In the multicore approach, Xtratum virtualizes all available physical CPUs offering to partitions vCPUs (virtual CPUs). A partition is capable of using one or more vCPUs to execute the code inside partitions. In this version, which is compatible with ARM processors, the maximum number of vCPUs was 2. This experiment proves that multicore approaches offer better performances for communication between partitions in Xtratum.

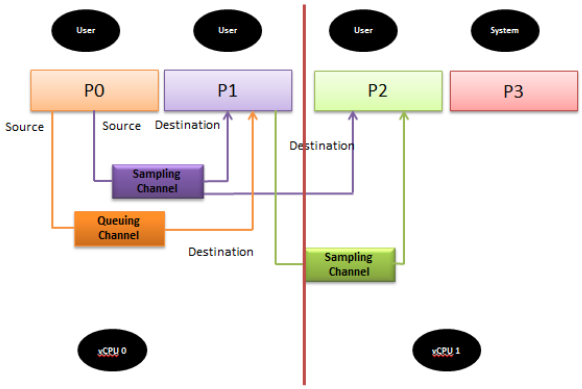

Fig. 10 Multi core test architecture.

These tests were executed 10 times to get a stable value. Different scheduling policies were also tested but the one followed in this implementation seemed to be more stable where no messages were lost. Different frequencies were tested as well on Xtratum, but in some cases it did not influence the results only in the case 600 MHz where some messages were lost. At first different emplacement of partitions was used, putting P0 and P2 for example in one core or P1 and P2 in one core but messages were not received so the emplacement used was the most stable and messages were all received.

IV. CONCLUSION

Virtualization offers multiple advantages for embedded systems especially for time critical systems. Even though hypervisors used for embedded systems are more likely to be much optimized to adhere to critical schedules requirements, their performance must not be affected. Several modern embedded systems are complex computing entities that execute critical tasks, and security in these devices is a critical task. In this paper the importance of self-diagnostics applications on the hardware level and also on the software layer was emphasized. A performance analysis of inter partitions channels found in Xtratum on a cortex A9 processor was provided. Without effective self-diagnostics, designed and configured for the Mixed Criticality Systems, any undetected hardware failure can reduce the system reliability with and impact on safety and security. In the future, a General Purpose Operating System (GPOS) will be ported on Xtratum with Authentication and Access Control applications.

REFERENCES

[1] J. Jung, I. Ahmed, "Development of Field Programmable Gate Array-based Reactor Trip Functions Using Systems Engineering Approach," Nuclear Engineering and Technology, Volume 48, Issue 4, 2016.

[2] Tellabi, I. Ben Zid, C. Ruland and K. Waedt, "Virtualization on Secure Platforms for Industrial Applications Current use cases and future perspectives,".12th International Conference on Reliability Maintainability and Safety (ICRMS), October, 2018.

[3] Mixed-Criticality Forum, "Mixed-Criticality Forum," Website, 2017, last visited on 16/01/2019. [Online]. Available: http://www.mixedcriticalityforum.org/.

[4] IAEA Nuclear Security Series No. 13, "Technical Challenges in the Application and Licensing of Digital Instrumentation and Control Systems in Nuclear Power Plants," IAEA, 2015.

[5] A. Tellabi, Y. Sassmanhausen, E. Bajramovic, and C. Ruland, " Overview of Authentication and Access Controls for I&C systems," IEEE 16th international conference on industrial informatics, 2018.

[6] Y. Zhao, Z. Yang, and D. Ma , "A survey on formal specification and verification of separation kernels,". Front. Comput. Sci. 11, 585-607, 2017. DOI: https://doi.org/10.1007/s11704-016-4226-2.

[7] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench and J. Nowotsch, "Mixed-Criticality Embedded Systems – A Balance Ensuring Partitioning and Performance," in Digital System Design (DSD), 2015 Euromicro Conference on, Aug 2015. doi: 10.1109/DSD.2015.100 pp. 453–461.

[8] Y. Zaki, "An Embedded Multi-Core Platform for Mixed-Criticality Systems Study and Analysis of Virtualization Techniques,",2017.

[9] A. Tellabi. L. Peters, C. Ruland and K. Waedt, "Security Aspects of Hardware Virtualization Technologies for Industrial Automation and Control Systems ,". GIACM WS on I4.0/IACS Standardization, Berlin, September 2018.

[10] Xilinx, "Xilinx Support Forum," Website, 2017, last visited on 10/02/2019. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/zc706/14_5/zc706-bist-pdf-xtp242-14.5-c.pdf.

[11] Fentiss, "Fent Innovative Software Solutions," last visited on: 4/03/2019 [online]. Available: http://www.fentiss.com/.

[12] M. Masmano, I. Ripoll, A. Crespo, and J.J. Metge, "Xtratum: a Hypervisor for Safety Critical Embedded Systems," in 11th Real-Time Linux Workshop. Citeseer, 2009, pp. 263-272.

[13] Xtratum , “Xtratum Hypervisor for ARM,”last visited on: 4/01/2019 [online]. Available: http: http://www.fentiss.com/.
[14] S. Pinto, J. Pereira, T. Gomes, M. Ekpanyapong, and A. Tavares “Towards a TrustZone-assisted Hypervisor for Real Time Embedded Systems,” in IEEE computer architecture letters , 2016.
[15] Mixed-Criticality Forum, “Mixed-Criticality Forum,” Website, 2017, last visited on 03/02/2019. [Online]. Available: http://www.mixedcriticalityforum.org/.
[16] T. Koller, G. Gala, D. Gracia Pérez, C. Ruland, and G. Fohler, “Dreams: Secure communication between resource management components in networked multi-core systems,” in 2016 IEEE Conference on Open Systems (ICOS), IEEE Conference on Open Systems (ICOS). IEEE Computer Society, October 2016, pp. 99–104, Best Paper Award.
[17] DREAMS: Distributed REal-time Architecture for Mixed Criticality Systems,” Website, 2017, last visited on 11/04/2019. [Online]. Available: http://dreams-project.eu .
[18] OVERSEE Consortium, “OVERSEE: Open Vehicular Secure Platform,” Website, 2012, last visited on 30/01/2019. [Online]. Available: https://www.oversee-project.com/.

Asmaa Tellabi was born in Morocco in 1993. She received her B.S. degree in software engineering from the International University of Rabat, Rabat, Morocco, in 2014 and the M.S. degree in IT Security from the International University of Rabat University, Rabat, Morocco, in 2016. She is currently pursuing the Ph.D. degree in cybersecurity for critical infrastructures at Framatome GmbH and University of Siegen, Germany.
From 2017, she has been a PhD candidate at Framatome GmbH, Erlangen, Germany. Her research interest includes the development of new secure platforms based on virtualization techniques for critical infrastructures, security controls for instrumentation and control systems used in Nuclear Power Plants, and hardware security.

Karl Christoph Ruland studied mathematics, computer science and physics. He received the Diploma and Dr.-degree in mathematics at the University of Bonn, Germany. After 6 years in the industry he became professor for Data Communications at the University for Applied Sciences, Aachen, Germany, in 1982. Then, a full time professor at the University of Siegen, Germany in 1992. His research focus is the integration of cryptography based security into communications systems, preferably real-time and industrial oriented systems. He served as Co-Chair of the Security WG of the eSafety Forum (for automotive security and safety) of the EU commission. For the last years he specialized in Smart Grids control security.

Dr. Karl Waedt got his M.S in Computer Science from Univ. Erlangen-Nürnberg in 1990. He started working at Siemens KWU on Design & Software Development of TXS. In 1994, he got his PhD in Computer Science (Responsive Distributed Systems). In 2002 he became the project leader of TELEPERM XS FUTIS Software Development. In 2004, he became the TXS System Software Development Section Head and in 2006 he specialized in Cybersecurity aspects for TXS Platform. He participated in Cybersecurity for OL3 and German I&C projects, overall I&C / IT Security Cybersecurity Section Head. In 2015, he coordinated Cybersecurity in I&C & ES Projects of Areva GmbH, Management of the partially German BMWi Ministry funded SMARTEST-AREVA Cybersecurity R&D Project (until end 2019, 8 PhD students, 6 partner Universities). He is the Deputy Chair of DKE UK 967.1 (German Mirror Committee of TC45/SC45A), German Delegate in TC45/SC45A WG3 (I&C) and WG9 (Cybersecurity), Chairperson of CEN/CENELEC CLC/TC 45AX (I&C), Deputy in KTA UK-EL (I&C and ES Board); Contributor to IAEA TMs on Cybersecurity, on behalf of German Ministry BMWi, Member of DKE/TBINK Safety & Security by Design, German Delegate in ISO/IEC JTC1/SC27 WG4 Security Controls and Services, Member of GI (German Informatics Society) and IEEE.

Sabri Abdelbast was born in Youssoufia, Morocco. He got his BS at university of Paderborn in 2016. Currently he is writing his master thesis at Framatome GmbH and university of Erlangen-Nürnberg, Germany. His focus is on microelectronics and security controls on the hardware level.