embeddedworld2023
Exhibition&Conference
... it´s a smarter world

Incorporating CI/CD workflow allowing remote interactions with hypervisor enabled MPSoC

The XANDAR project use case

C. Panagiotou, D. Karadimas
AVN Techgroup,
Limassol, Cyprus
{chpanag, karadimas}@avntechgourp.com

W. Zaeske, P. Nöldeke
Institute of Flight Systems,
German Aerospace Center (DLR), Brunswick, Germany
{Wanja.Zaeske, Phillip.Noeldeke}@dlr.de

N. Petrellis, G. Keramidas, C. Antonopoulos, N. Voros
University of Peloponnese, Patras, Greece
npetrellis@go.uop.gr, {g.keramidas, c.antonop, voros}@esdalab.ece.uop.gr

V. Morales-Mora, M. Masmano-Tello, V. Nicolau-Gallego
Fent Innovative Software Solutions (fentISS), SL, Spain
{vmorales, mmasmano, vnicolau}@fentiss.com

J. Becker,
Karlsruhe Institute of Technology (KIT), Germany,
juergen.becker@kit.edu

***Abstract*— The adoption of more efficient development methodologies is rapidly turning into an absolute imperative for embedded systems due to market demands for ever more complex systems and faster development cycles. This is particularly true for safety critical embedded systems in which rapid prototyping and development is required while key qualities like trustworthiness and safety should be guaranteed. In this context, organizations need solutions supported by embedded DevOps processes leveraging efficient methodologies effectively addressing respective shortcomings by traditional embedded development methods. The XANDAR (EU funded project) toolchain incorporates an end-to-end Continuous Integration/Continuous Development (CI/CD) process for building application software components based on the XtratuM (XNG), a bare-metal space-qualified hypervisor aimed at safe and efficient embedded real-time systems. The pipeline of the fully functional, proposed CI/CD approach paves the path to facilitate the deployment of the XNG-based applications intoMulti Processor System on Chips (MPSoC) where XNG is integrated. The operation of the applications is monitored and evaluated against predefined test vectors that the automation server consumes in order to decide whether the application requirements are met.**

Keywords—XNG, XtratuM, X-by-Construction, safety-critical, CI/CD, MPSoC

I. Introduction

Designing embedded systems for safety-critical applications, such as the aerospace or the automotive domains is an inherently challenging task. This is especially true if safety requirements are combined with properties, such as the need for high computational performance, the integration of non-deterministic Machine Learning (ML) algorithms, or external communication interfaces that connect the system to an untrusted network. In these cases, an appropriate combination of safety mechanisms must be applied to prevent the occurrence of hazardous events. As the complexity of embedded systems grows, this process can become a time-consuming and error-prone endeavour.

In that context, the XANDAR[1] project aims to increase the degree of automation that is associated with this complex task. The devised approach is based on an extensible library of safety patterns, i.e., a collection of verified design-time procedures for safety that can be annotated to certain system entities. In the sense of the X-by-Construction (XbC) paradigm, the design-time procedures associated with annotated patterns are executed to auto-generate hardware and software artefacts that implement the desired safety mechanisms.

A common situation in the systems development process is to have plenty of different applications with their own safety and security requirements. Moreover, the application integration with various critical levels is very common. The issue in both cases is the same, i.e., the error propagation. It is important to ensure that an unsafe part of the system will not spread any kind of error to the critical zone. The approach to avoid those problems is the time-space partitioning.

In that context, the XtratuM Next Generation (XNG) hypervisor[2] is employed in the XANDAR toolchain. The target is to provide a partitioned solution, using para-virtualization. XNG allows mixed criticality implementations

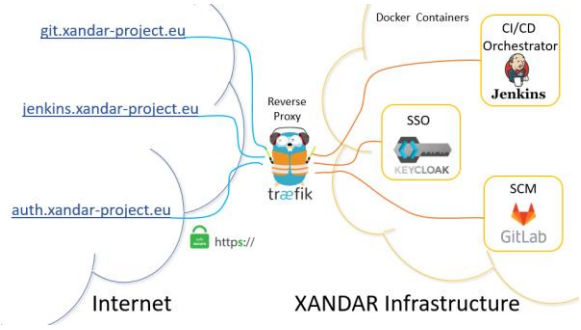

Figure 2 XANDAR CI/CD Infrastructure

through MMU/MPU mechanisms, and fixed cyclic scheduler for spatial and temporal isolation respectively.

Considering the high complexity of such systems, the heterogeneity of the involved components, and their tolerance in errors, the use of mechanisms that increase the trust to the final system even from design time becomes imperative. This is where Continuous Integration / Continuous Delivery techniques (CI/CD) come into play. CI/CD practices are currently the most widely accepted choice to shorten software development and delivery cycle time without degrading quality assurance. Employing CI/CD within the XANDAR project ensures compatibility between the individual components of the toolchain and considerably improves the quality of the XANDAR toolchain.

This paper presents the realization of state-of-the-art CI/CD techniques for building, deploying, and tracing applications for XNG-based MPSoCs. Section II presents the CI/CD tools and infrastructure adopted in the XANDAR project. Section III presents the XNG architecture that is used as the execution environment of the applications. Section IV describes the workflow that was automated and integrated to the infrastructure as a CI/CD pipeline. Finally, the remarks and conclusions of this work are presented in Section V.

II. XANDAR CI/CD INFRASTRUCTURE

It is important to mention that in order to perform automated tests of toolchain components, relevant tools must be integrated into the CI/CD framework in such a manner that they can be automatically executed and applied to scenarios defined by the test cases. The main tools that are involved in this process are related to source code management (SCM) and orchestration of the automation processes.

In XANDAR, git[3] has been adopted as the SCM tool (supported by a self-hosted Gitlab platform) [4], a distributed version-control system for tracking changes in any set of files.

The automation server, responsible to orchestrate all the required tasks is Jenkins [5] with its support for the so-called Jenkins pipelines. Jenkins pipeline is a suite of plugins which supports the implementation and integration of continuous delivery pipelines into Jenkins. The pipeline provides an extensible set of tools for modelling simple-to-complex delivery pipelines "as code" via the pipeline domain-specific language (DSL) syntax. The definition of a Jenkins pipeline is written into

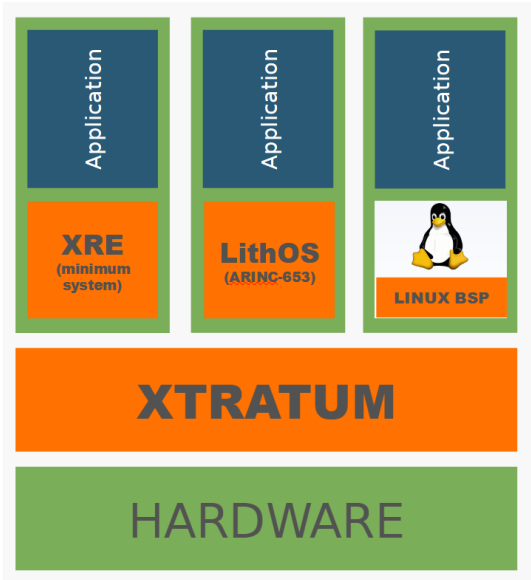

Figure 1 Guest application kernels in XANDAR

a text file (called a Jenkinsfile) which in turn can be committed to a project source control repository.

The overall architecture of the XANDAR CI/CD infrastructure along with the supported auxiliary tools (single sign on, reverse proxy, etc.) are presented in Figure 1.

III. XNG ARCHITECTURE

XtratuM Next Generation (XNG) [2] hypervisor is a type 1 embedded hypervisor that is currently qualified under ECSS category B in the SPARC v8 and ARM v7A/R architectures.

XNG provides the services needed by safety-critical systems, such as hypervisor and partition management, support for normal and privileged partitions, resource virtualization through the Partition Virtual Execution Environment (PVEE), temporal partitioning, spatial partitioning, inter-partition communication (IPC) mechanisms, a Health Monitor (HM) service which detects faults in the hardware and in XNG, observability (traces and debug) of the system, and the XNG Configuration Files (XCF), which are a set of XML files to allow the system integrator to configure the system.

In XtratuM, each application software is executed in a time and space segregated virtual execution environment known as partition. The segregation in space is achieved by using the available hardware mechanisms such as the Memory Protection Unit (MPU) or the Memory Management Unit (MMU)). Partitions are scheduled according to a cyclic scheduling policy enforcing the temporal isolation.

XNG hypervisor paravirtualizes parts of the hardware platform to partitions, which means that the guest application kernel needs to be modified to be executed inside a partition. This way XNG is able to delegate I/O devices and interrupts to partitions. As shown in Figure 2, in the context of XANDAR project, three guest application kernels have been created:

- XRE partition, as a minimal runtime environment.
- Linux-based partition.
- LithOS-based partition.

A native single core or multicore application can be executed in a partition thanks to the XtratuM Runtime Environment (XRE), which allows the execution of bare metal applications over XNG without the need of any guest OS. XNG provides services in order to use the hypervisor capabilities as communication ports, virtual interrupts, or hardware access. The bare metal application can be developed in C language.

The Linux-based partitions use a patched version of embedded-Linux. The system file, the device drivers, and other Linux capabilities are available to the partition with this implementation.

LithOS[6] is a real-time operating system (RTOS) designed specifically to be executed inside an XNG partition in an Integrated Modular Avionics (IMA) -based system supporting space flight functions of a criticality classification of up to category B according to European Cooperation for Space Standardization (ECSS).

Its purpose is to act as a guest real-time operating system providing services compliant with the APEX API defined by the ARINC-653 P1 specification [7] and a subset of P2[8]. In the terminology used by the ARINC-653, LithOS and XNG should jointly be considered as the core software of an IMA[7]. Some of the main features of LithOS are:

- Configuration of the maximum number of resources that will be allocated to support the services provided by its ARINC-653 P1 API[7]
- Partition management services that aim at controlling the internal state of its own partition or other partitions
- Execution threads (called processes) which execute concurrently with other processes of the same partition in a periodic or an aperiodic way.
- Time control for process management by using the global time provided by the virtualization layer (i.e., XNG)
- Intra-partition communication and synchronization mechanisms
- Inter-partition communication services
- A Health Monitor (HM) service inspired in the ARINC-653 HM[7]
- Multiple module schedule policies as defined in[8]
- Hardware interrupt management
- System management services

IV. CI/CD WORKFLOW

The CI/CD environment orchestrates the whole evaluation process of an application, starting from its modeling, down to its deployment on the target platform and the assessment of its output. The application is modeled at the top level as one or more Directed Acyclic Graphs (DAG). The nodes of a DAC are dependent tasks that can be hierarchically analyzed to runnable subtasks. Multilevel scheduling approaches such as Earliest Deadline First (EDF) are tested at the top level to ensure that the local and global deadlines are met for each task. Multiple application DAGs are generated automatically to test the efficiency of the underlying scheduling and deployment mechanisms.

To evaluate the correctness of a scheduling plan, the CI/CD environment deploys the application on a target platform through the XNG hypervisor. The DAG tasks are implemented as XNG partitions and the XNG scheduling mechanisms are employed to realize the scheduling that has been determined in high level. The physical processor cores of the target platform are used to execute safely parallel tasks. The target platform can be one of the multicore FPGAs supported by XNG, like Xilinx Zynq-7000 or ZynqMP Ultrascale+.

The baseline of XNG based applications is the XNG configuration which defines the partitions, their schedule, their communication channels, memory allocation and the health monitor. Communication channels can either implement Inter Partition Communication (IPC) or enable access to external input/output interfaces.

As already mentioned in the previous section, the description of the configuration is set up in several Extensible Markup Language (XML) files. The idea of the XANDAR toolchain is to export the properties of a hypervisor configuration from a model description (e.g., SysML or DAG) and autogenerate these XML files by merging fixed XML code with application and task-dependent XML descriptions. The actual functions and applications are also part of this model description and denoted as software components (SWC). For each SWC, all properties are derived to embed the SWC into a partition.

Within this process, skeleton code for the SWC is generated, which allows an application developer to focus on implementing the software of the SWC. In the meantime, the toolchain takes care of glue code and XNG hypervisor integration. The glue code includes an abstraction over the previously defined IPC channels and a trigger function for an event-based execution of SWCs. Moreover, (de-)serialization code is derived from the messages defined in the toolchains input.

Now that all required artifacts for a complete XNG configuration are available, the next step is the generation of a bootable system image with the supervision of the CI/CD environment. The system image includes both the SWC's compiled code and the configuration including schedule and configuration channels. There are three different ways to deploy the resulting system image into the target hardware. First, the image can be directly loaded by the CI/CD environment into Random-Access-Memory (RAM) and executed for testing using Joint Test Action Group (JTAG). The second way integrates the image into a standalone bootable artifact that can be loaded from a Secure Digital (SD) card. However, this method cannot be automatically performed by the CI/CD environment without human intervention. The third option is a combination of a first and second stage bootloader stored into the onboard flash memory to enable the deployment of bootable images via a laboratory network which is suitable for larger system images.

This workflow is implemented as a pipeline in the Jenkins automation server to facilitate the automated and unsupervised execution and primary evaluation of applications running on a

target platform. XNG provides plenty of services to partitions. In that context, some of those services have been chosen to summarize the XNG performance. The performance tests have been executed by an application running over an XRE partition in the scope of an ECSS category B qualification process of the hypervisor. Performance has been evaluated with respect to the mean time of the following services:

- Get system time service provides the current system time in μseconds
- In port: Read from a hardware I/O port
- Reset partition: Restart a specific partition
- Set system timer: Trigger a virtual timer using the system clock as reference

In addition, three different cache configurations can be applied and evaluated:

- Non–cacheable (NC)
- Writeback (WB)
- Write through

The performance for the described cache allocation policies is presented in the following tables (TABLE I. to TABLE V.

TABLE I. GET SYSTEM TIME SERVICE METRICS IN MICROSECONDS

Cache config.	Mean Time Cost (μs)
NC	53.49
WB	10.07
WT	11.24

TABLE II. IN PORT SERVICE METRICS IN MICROSECONDS

Cache config.	Mean Time Cost (μs)
NC	48.10
WB	9.06
WT	11.41

TABLE III. RESET PARTITION SERVICE METRICS IN MICROSECONDS

Cache config.	Mean Time Cost (μs)
NC	1588.06
WB	620.98
WT	621.94

TABLE IV. SET SYSTEM TIMER SERVICE METRICS IN MICROSECONDS

Cache config.	Mean Time Cost (μs)
NC	35.22
WB	5.45
WT	7.93

A typical overhead that exists at partitioned systems is the context switch overhead. The following table shows the average context switch overhead in specific cases with different cache configurations:

TABLE V. CONTEXT SWITCH METRICS IN MICROSECONDS

Test case	Cache config	Mean context switch (µs)
Between partitions	NC	1094.65
	WB	594.40
	WT	178.30
Between partition and idle	NC	169.80
	WB	33.25
	WT	3460

The presentd performance evaluation results are collected on a Zynq-7000 FPGA platform with XNG 1.4 assuming the following configuration:

TABLE VI. CONFIGURATION OF THE ZYNQ-7000

Property	**Configuration**
CPU	Frequency 500 MHz
Cache configuration	Instruction Cache: enabled. Data Cache: enabled. Memory areas cache policy • NC: Non-Cacheable • WT: Write-through • WB: Write-back.
Error Correction Checking (ECC)	Enabled

V. CONCLUSION

The complexity of designing and implementing embedded systems for safety – critical systems is a determinant factor for integrating such systems in domains with increased challenges in safety and security (e.g. aviation, automotive). The incorporation of CI/CD techniques that automate the integration and testing processes can speed up the delivery time of reliable solutions. This paper presented the XANDAR approach in CI/CD and described the incorporation of a workflow that allows remote interactions with hypervisor enabled MPSoC. This work facilitates the deployment and evaluation of XNG - based applications executed on the target system.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 957210.

REFERENCES

[1] https://xandar-project.eu/
[2] https://fentiss.com/products/hypervisor/
[3] https://git-scm.com/
[4] https://about.gitlab.com/
[5] https://www.jenkins.io/
[6] https://www.fentiss.com/lithos/
[7] Aeronautic Radio, Inc. Avionics Application Software Standard Interface. Part 1 - Required Services. ARINC Specification 653P1-3, 2010.
[8] Aeronautic Radio, Inc. Avionics Application Software Standard Interface. Part 2 - Extended Services. ARINC Specification 653P2-2, 2010”