MASTER THESIS

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Mechatronics/Robotics

Virtualisierung eines Echtzeit-Betriebssystems zur Steuerung eines Roboters mit Schwerpunkt auf die Einhaltung der Echtzeit

By: Halil Pamuk, BSc

Student Number: 51842568

Supervisor: Sebastian Rauh, MSc. BEng

Wien, April 15, 2024

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Wien, April 15, 2024 Signature

Kurzfassung

Erstellung einer Echtzeit-Robotersteuerungsplattform unter Verwendung von Salamander OS, Xenomai, QEMU und PCV-521 in der Yocto-Umgebung. Die Plattform basiert auf Salamander OS und nutzt Xenomai für Echtzeit- Funktionen. Dazu muss im ersten Schritt die Virtualisierungsplattform evaluiert werden. (QEMU, Hyper-V, Virtual Box, etc.) Als weiterer Schritt folgt die Anbindung eines Roboters über eine VARAN-Bus Schnittstelle. Das gesamte System wird in der Yocto-Umgebung erstellt und konfiguriert. Das Hauptziel der Arbeit ist es, herauszufinden, wie die Integration von Echtzeit-Funktionen und effizienten Kommunikationssystemen in eine Robotersteuerungsplattform die Reaktionszeit und Zuverlässigkeit von Roboteranwendungen verbessern kann

Schlagworte: Schlagwort1, Schlagwort2, Schlagwort3, Schlagwort4

Abstract

Abstract

Keywords: Echtzeit, Virtualisierung, Xenomai, VARAN

Contents

1 Introduction

In today's industrial production and automation, robot systems are well established and of crucial importance. Robots must react to their environment and perform time-critical tasks within strict time constraints. Delays or errors can have catastrophic consequences in some cases. Traditional operating systems, such as Windows or Linux, are often not suitable for these types of real-time requirements as they cannot guarantee deterministic execution times. Therefore, real-time operating systems are required that are specifically designed to react to events within fixed time limits and prioritise the execution of high-priority processes.

The core component of an RTOS that enables real-time capabilities is the kernel. The kernel is responsible for managing system resources, scheduling tasks, and ensuring deterministic behavior. It employs preemptive scheduling mechanisms to allow high-priority tasks to preempt lower-priority tasks, ensuring that time-critical tasks are not delayed. The kernel also implements priority-based scheduling algorithms, such as Rate Monotonic Scheduling (RMS) or Earliest Deadline First (EDF), to schedule tasks based on their priorities and timing constraints. Additionally, RTOS kernels are designed to minimize interrupt latency, which is crucial for real-time applications that require immediate response to external events.

In these RTOS systems, task scheduling is based on so-called priority-based preemptive scheduling. Each task in a software application is assigned a priority. A higher priority means that a faster response is required. Preemptive task scheduling ensures a very fast response. Preemptive means that the scheduler can stop a currently running task at any point if it recognizes that another task needs to be executed immediately. The basic rule on which priority-based preemptive scheduling is based is that the task with the highest priority that is ready to run is always the task that must be executed. So if both a task with a lower priority and a task with a higher priority are ready to run, the scheduler ensures that the task with the higher priority runs first. The lower priority task is only executed once the higher priority task has been processed. Real-time systems are usually categorized as either soft or hard real-time systems. The difference lies exclusively in the consequences of a violation of the time limits.

Hard real-time is when the system stops operating if a deadline is missed, which can have catastrophic consequences. Soft real-time exists when a system continues to function even if it cannot perform the tasks within a specified time. If the system has missed the deadline, this has no critical consequences. The system continues to run, although it does so with undesirably lower output quality.

1.1 Application Context

This master's thesis was written at SIGMATEK GmbH & Co KG [1]. SIGMATEK uses its own customized Linux distribution to be run on their self-manufactured CPUs, namely Salamander 4. This operating system employs hard real-time with latency requirements between 20 and 50 µs. The goal is to virtualize Salamander 4 and approach the performance of bare metal CPUs. Salamander 4 is virtualized through a third party service, QEMU. The details of this operating system are explained in chapter 3.

1.2 State of the art

1.3 Problem and task definition

1.4 Objective

The main objective of this work is to create a real-time robot control platform that integrates Salamander OS, Xenomai, QEMU and PCV-521 in the Yocto environment.

2 Methodology

This section describes in detail all the theoretical concepts and boundary conditions as well as practical methods that contributed to achieving the objectives of this master's thesis.

Trace-cmd was used for tracing the Linux kernel. It can record various kernel events such as interrupts, scheduler decisions, file system activity, function calls in real time. Trace-cmd helped in getting detailed insights into system behaviour and identify reasons for latency [2].

The data that was recorded by trace-cmd was then fed into Kernelshark, which is a graphical front-end tool [3]. It visualizes the recorded kernel trace data in a readable way on an interactive timeline, which facilitated the process of identifying patterns and correlations between events. By further filtering the displayed events according to specific criteria such as processes, event types or time ranges, the latency issues were analyzed.

Real-time operating system capabilities were provided by Xenomai, which is real-time development framework that extends the Linux kernel. It enables low-latency and deterministic execution of time-critical tasks. Xenomai introduces a dual-kernel approach with a real-time kernel coexisting alongside Linux. A key utility within the Xenomai suite is the latency tool, which benchmarks the timer latency - the time it takes for the kernel to respond to timer interrupts or task activations. The tool creates real-time tasks or interrupt handlers and measures the latency between expected and actual execution times [4].

3 Salamander 4

Salamander 4 is the proprietary operating system of SIGMATEK. It is based on Linux version 5.15.94 and integrates Xenomai 3.2, a real-time development environment [4]. Salamander 4 is a 64-bit system, which refers to the x86_64 architecture. The real-time behaviour is achieved through the use of Symmetric Multi-Processing (SMP) and Preemptive Scheduling (PREEMPT). In addition, it uses IRQPIPE to process interrupts in a way that meets the real-time requirements of the system. The output of the command `uname -a` can be observed in code 1.

```
root@sigmatek-core2:~# uname -a
Linux sigmatek-core2 5.15.94 #1 SMP PREEMPT IRQPIPE Tue Feb 14 18:18:05 UTC
    2023 x86_64 GNU/Linux
```

Code 1: System information

Xenomai consists of 3 parts. These can be found in the Table 1.

Table 1: Xenomai architecture

Teil	Beschreibung
i-pipe	Kernelerweiterung für das Domain-Konzept
Xenomai Kernel	Benutzt die i-pipe, und hängt sich als root-Domain ein
Xenomai User	Programme (LRT) verwenden diese Bibliothek, um Xenomai Funktionen verwenden zu können.

Xenomai beruht auf einem Domain-Konzept, das bedeutet, dass alle IRQ an die erste Domain gesendet werden. (root – Domain / Xenomai) Nur wenn diese nichts mehr zu tun hat, dann darf die 2. Domain arbeiten. Das bedeutet, erst wenn alle Xenomai Task in einem Wartezustand sind, arbeiten die Linux-Tasks.

In der Regel unterbricht der Prozessor beim IRQ-Handling seine aktuellen Aktivitäten, um einen Interrupt zu bearbeiten, während die IRQ-Behandlung von Xenomai einen Interrupt-Pipeline-Mechanismus verwendet, der das gleichzeitige Abrufen und Vorbereiten eines anderen Interrupts ermöglicht, während ein Interrupt bearbeitet wird, was die Leistung verbessert und die Latenzzeit verringert.

Was Xenomai4 von seinem Vorgänger Xenomai3 unterscheidet, ist die vollständige Neugestaltung der Ausführungsphase mit hoher Priorität. Dies geschah aus Gründen der Portabilität und

Wartungsfreundlichkeit: I-pipe - die zweite Iteration der ursprünglichen Adeos-Interrupt-Pipeline - wurde vollständig durch Dovetail ersetzt.

Table 2: Domain specific functions

Xenomai spezifische Funktionen	**Linux spezifische Funktionen**
Tasks	Dateizugriffe
Mutexes, Semaphoren, Events	Netzwerk

Ein Aufruf dieser Funktionen erfordert die entsprechende Domain. Wenn der Task in der falschen Domain läuft, dann wird ein Domain-Wechsel forciert. Ein Domainwechsel von Xenomai nach Linux geht relativ einfach. Aber der Wechsel von Linux nach Xenomai braucht Unterstützung, und dafür ist die Hilfe des Gatekeepers notwendig. Das bedeutet, der Gatekeeper hilft einem Task von Linux nach Xenomai zu wechseln.

3.1 Salamander 4 Description

3.1.1 Task priorities

Es gibt grundsätzlich 4 Gruppen

Table 3: Overview of the priority groups and their relationships

Prioritätsgruppe	**Bereich**
Xenomai Priorität	0 bis 99
Linux RT Priorität	1 bis 99
Linux (Nice Level) Priorität	-20 bis 19
RTK Priorität	0 bis 14

3.1.2 Memory Management

Es gibt verschiedene Speicherbereiche

Linux/System/Programm Speicher Der Speicher, den Linux und Programme belegt haben. Dieser Speicher ist intern in viele Teile aufgeteilt. (DMA, . . .)

LRT-Heap Speicher Speicher den der LRT verwendet, oder welcher über ein CIL Funktionen angefordert wird.

App Heap, App Code, . . .

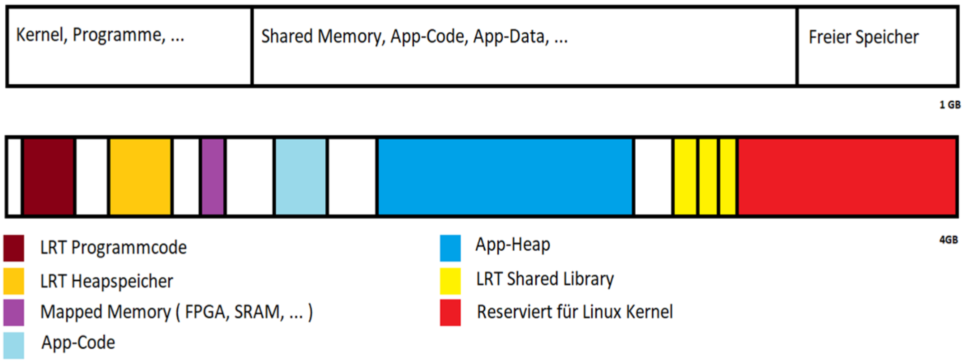

Figure 1: Memory Management

Eine LASAL CPU besteht aus den folgenden Software-Modulen:

- Operating system
- Loader
- Hardware-Klassen

Die Schnittstelle zwischen den einzelnen Modulen wird in Abbildung 2 durch einen Pfeil gekennzeichnet.

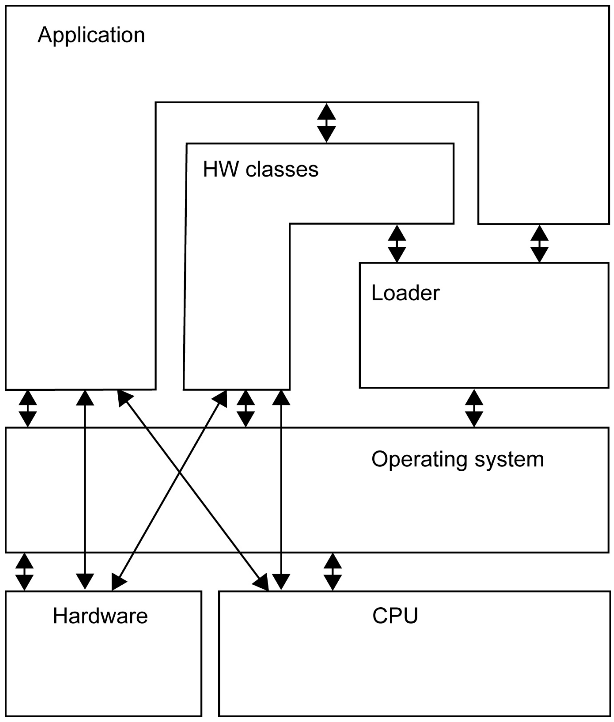

Figure 2: LASAL CPU

3.2 Salamander 4 Bare Metal

Salamander 4 Bare Metal refers to the proprietary hardware of SIGMATEK used to employ the custom operating system, including

Figure 3 shows latency of hardware Salamander4.

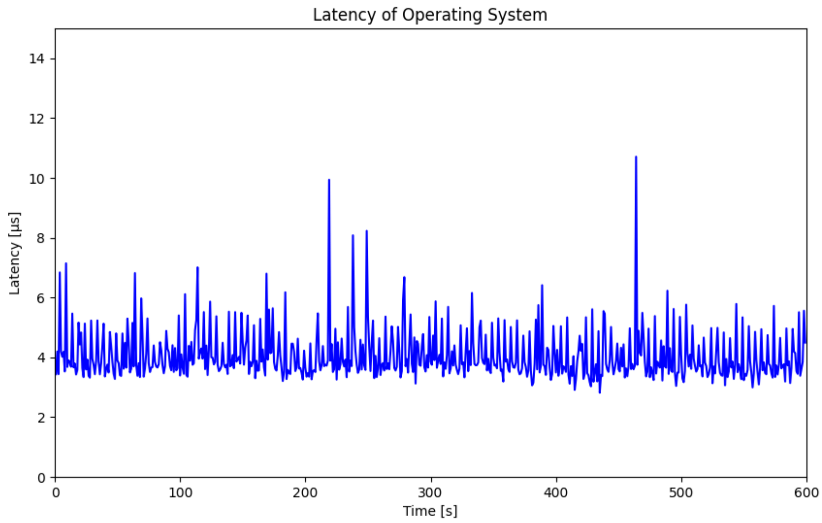

Figure 3: Latency hardware

3.3 Salamander 4 Virtualisation

In addition to providing Salamander 4 on its own hardware, SIGMATEK has also developed a virtualised version of this operating system. It was developped using Yocto, an open source project that allows customised Linux distributions to be created for embedded systems [5]. The virtualisation runs in a QEMU environment, which is an open source tool for hardware virtualisation [6]. With the help of the script depicted in code 3, Salamander 4 is started together with the necessary hardware components in the QEMU environment. This makes it possible to run Salamander 4 on a variety of host systems, regardless of the specific hardware of the host. Upon generating the necessary files, Yocto generates a QEMU folder with the following components shown in code 2.

```
sigma_ibo@localhost:~/Desktop/salamander-image$ ls -1
bzImage
drive-c
ovmf.code.qcow2
qemu_def.sh
salamander-image-sigmatek-core2.ext4
stek-drive-c-image-sigmatek-core2.tar.gz
vmlinux
```

Code 2: Contents of QEMU folder for Salamander 4

```
#!/bin/sh

if  [ ! -d drive-c/ ]; then
        echo "Filling drive-c/"
        mkdir drive-c/
        tar -C drive-c/ -xf stek-drive-c-image-sigmatek-core2.tar.gz
fi

exec  qemu-system-x86_64 -M pc,accel=kvm -kernel ./bzImage \
-m 2048 -drive
    file=salamander-image-sigmatek-core2.ext4,format=raw,media=disk \
-append "console=ttyS0 console=tty1 root=/dev/sda rw panic=1
    sigmatek_lrt.QEMU=1 ip=dhcp rootfstype=ext4 schedstats=enable" \
-net nic,model=e1000,netdev=e1000 -netdev bridge,id=e1000,br=nm-bridge \
-fsdev local,security_model=none,id=fsdev0,path=drive-c -device
    virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=/mnt/drive-C \
-device vhost-vsock-pci,guest-cid=3,id=vsock0 \
-drive if=pflash,format=qcow2,file=ovmf.code.qcow2 \
-no-reboot -nographic
```

Code 3: QEMU script for starting Salamander 4 virtualisation

Here is a description of the used components:

- **bzImage**: Compressed Linux kernel image, loaded by QEMU at system start.
- **ovmf.code.qcow2**: Firmware file for QEMU, enables UEFI boot process.
- **qemu_def.sh**: Shell script, starts QEMU with correct parameters to boot Salamamder 4 OS.
- **stek-drive-c-image-sigmatek-core2.tar.gz**: Archive containing files for C drive, unpacked and copied to drive-c/ directory by qemu_def.sh script.
- **drive-c**: Directory serving as C drive for QEMU system, created and filled by qemu_def.sh script.
- **salamander-image-sigmatek-core2.ext4**: Root file system for Salamander 4 OS, used as hard drive for QEMU system.
- **vmlinux**: Uncompressed Linux kernel image, typically used for debugging, contains debugging symbols not present in bzImage.

When the script is started from the host, the QEMU process can be scheduled to run on any available core, as it is noted bound to a specific CPU core. This means that the QEMU process may frequently switch between different cores, leading to an increase in latency. As the goal was to reduce latency in the guest, the first step was to isolate a CPU of the host and dedicate it solely to the QEMU process, so that it cannot be used for other tasks on user level. However, the isolcpus function only isolates at the user level and does not affect kernel tasks. Consequently, these kernel tasks and interrupts can still utilize the CPU.

Figure 4 shows latency of QEMU default Salamander4.

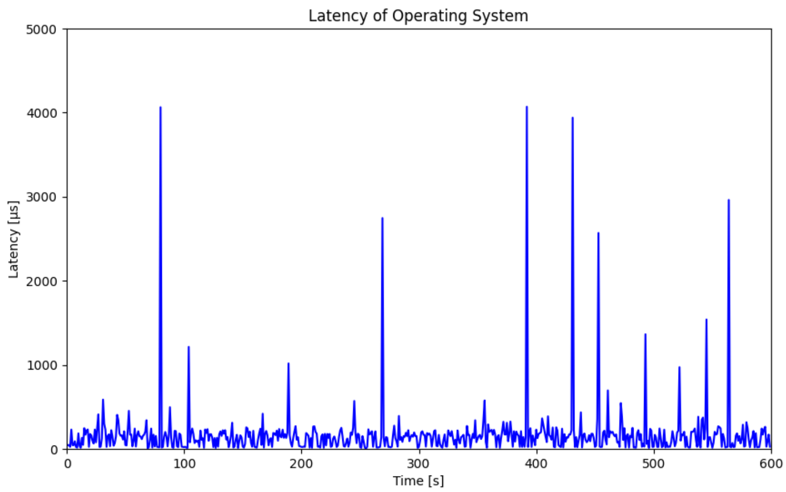

Figure 4: Latency no taskset

Figure 5 shows latency of QEMU taskset Salamander4.

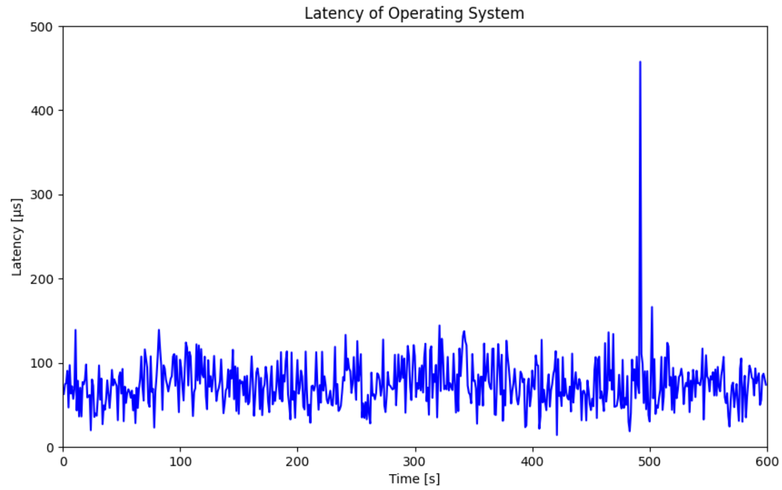

Figure 5: Latency taskset

Figure 6 shows latency of QEMU vapic Salamander4.

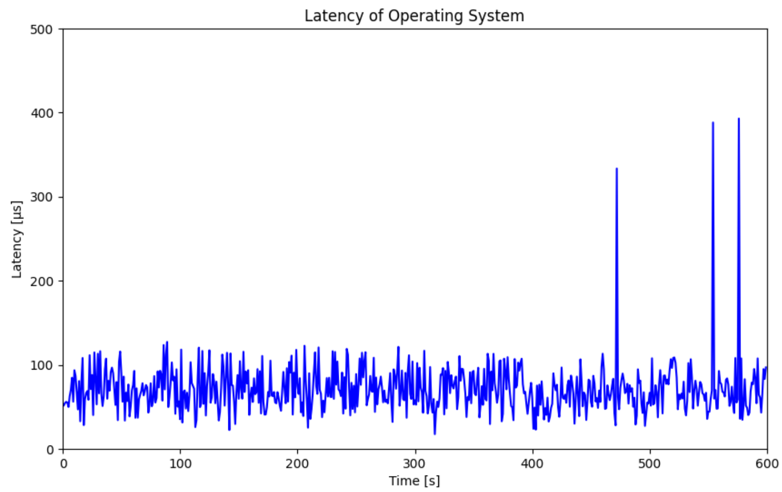

Figure 6: Latency taskset

Upon isolating a CPU to the QEMU process, it was anticipated that the guest would utilize nearly 100% of the CPU's capacity, with minimal to no intervention from the host. However, the isolcpus function only isolates at the user level and does not affect kernel tasks. Consequently, these kernel tasks and interrupts can still utilize the CPU. This led to the investigation of the causes for the observed high and inconsistent latency. The guest operates within the kvm_entry and `kvm_exit` events of the host. Kernelshark revealed a high frequency of `kvm_exit` events, indicating that the guest frequently relinquishes control of the CPU back to the host. This frequent switching hinders the guestś ability to run continuously, thereby increasing the virtualization latency. To further understand this, trace-cmd was employed to trace various events in the host-guest communication, including the reasons for these events. Specifically, the causes for `kvm_exit` events were analyzed. The command sudo trace-cmd record -e all -A @3:823 –name Salamander4 -e all was executed on the host for a duration of 5 seconds. The results in Figure 7 were obtained. Additionally, table 4 provides a short description of the observed `kvm_exit` events.

Exit Reason	**Description**
APIC_WRITE	Triggered when the guest writes to its APIC.
EXTERNAL_INTERRUPT	Triggered by external hardware interrupts.
HLT	Triggered when the guest executes the HLT instruction.
EPT_MISCONFIG	Triggered by a misconfiguration in the EPT.
PREEMPTION_TIMER	Triggered when the host's preemption timer expires.
PAUSE_INSTRUCTION	Triggered when the PAUSE instruction is executed.
EPT_VIOLATION	Triggered by a violation of the EPT permission settings.
IO_INSTRUCTION	Triggered when the guest executes an I/O instruction.
EOI_INDUCED	Triggered when an EOI signal is sent to the APIC.
MSR_READ	Triggered when the guest reads from a MSR.
CPUID	Triggered when the guest executes the CPUID instruction.

Table 4: Description of kvm_exit reasons

Figure 7 shows `kvm_exit` frequency with CPU islation.

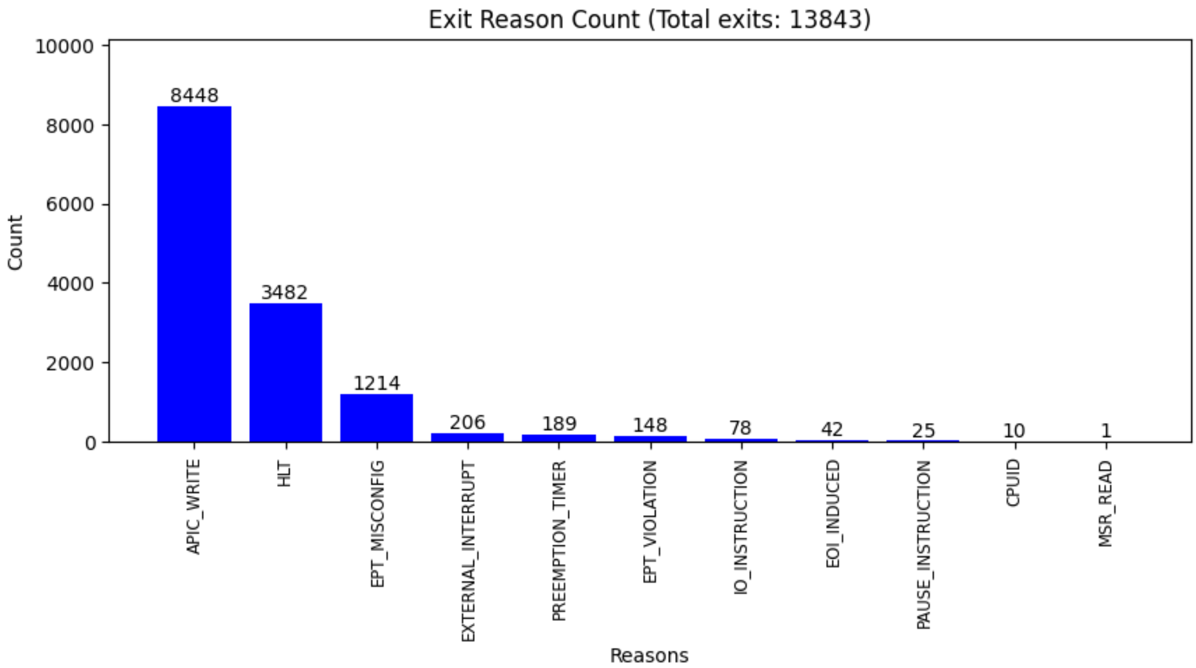

Figure 7: kvm exits

Figure 8 shows `kvm_exit` frequency without CPU islation.

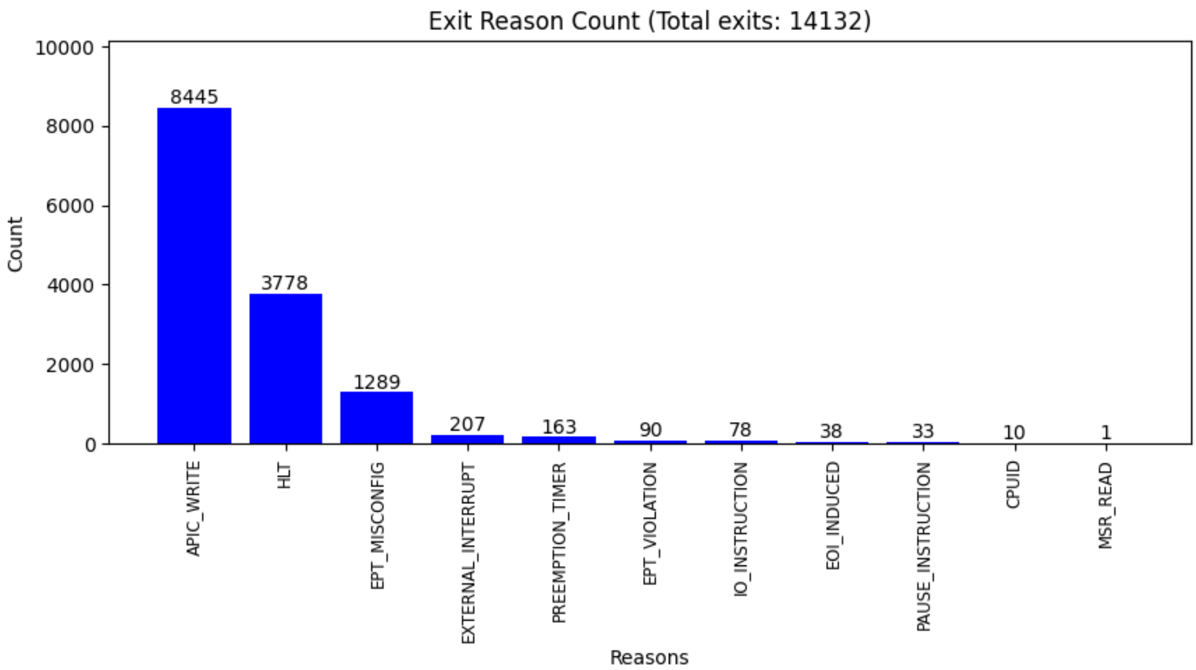

Figure 8: kvm exits default

Figure 9 shows `kvm_exit` frequency with APIC virtualisation.

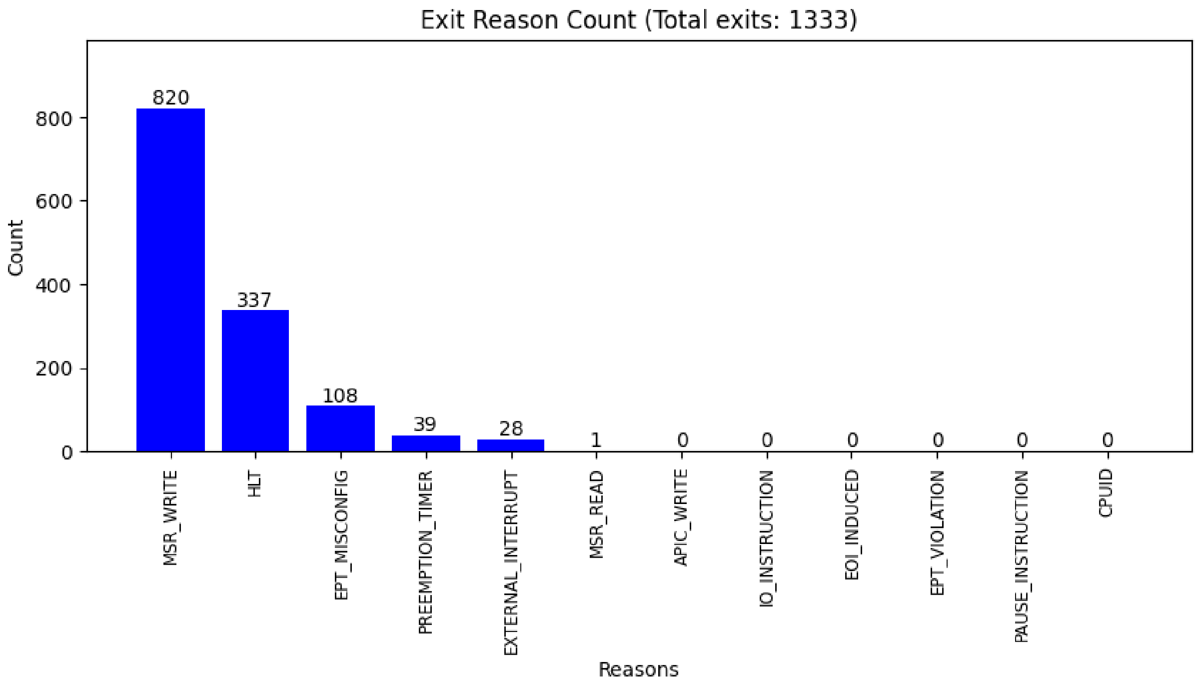

Figure 9: kvm exits default

Without CPU isolation, context switches take place at operating system level and not at hypervisor level. This explains why there are fewer `kvm_exit` events. However, this will also, as previously shown, lead to higher latency, as context switches at operating system level generally take longer than a `kvm_exit` and kvm_entry.
When the CPU was dedicated to the QEMU process, on the other hand,there was a significant increase in `kvm_exit` events. This is because every context switch takes place at hypervisor level. Nevertheless, lower latency was achieved thereby, as the qemu process is no longer influenced by the CPU scheduling of the operating system.
In the process of analyzing the `kvm_exit` events, several reasons for these exits were identified. The most frequent among these were the `APIC_WRITE` and `HLT` events. The former is initiated when the guest writes to its Advanced Programmable Interrupt Controller (APIC), a component of the CPU that manages hardware interrupts. The latter occurs when the guest executes the HLT instruction, effectively halting the CPU until the next external interrupt is fired. Other significant but less frequent events included `EXTERNAL_INTERRUPT` and `IO_INSTRUCTION`. These events are indicative of the guest's interaction with hardware devices and its execution of I/O operations. Events such as `EPT_MISCONFIG` and `PREEMPTION_TIMER` were also noted. These could potentially signal issues with memory management and the host's scheduling of the guest. While events like `PAUSE_INSTRUCTION`, `EPT_VIOLATION`, `EOI_INDUCED`, `MSR_READ`, and `CPUID` were the least frequent, they still provide valuable insights into the guest's behavior and the host-guest interaction.

The gnuplot latency is visible in Figure 10

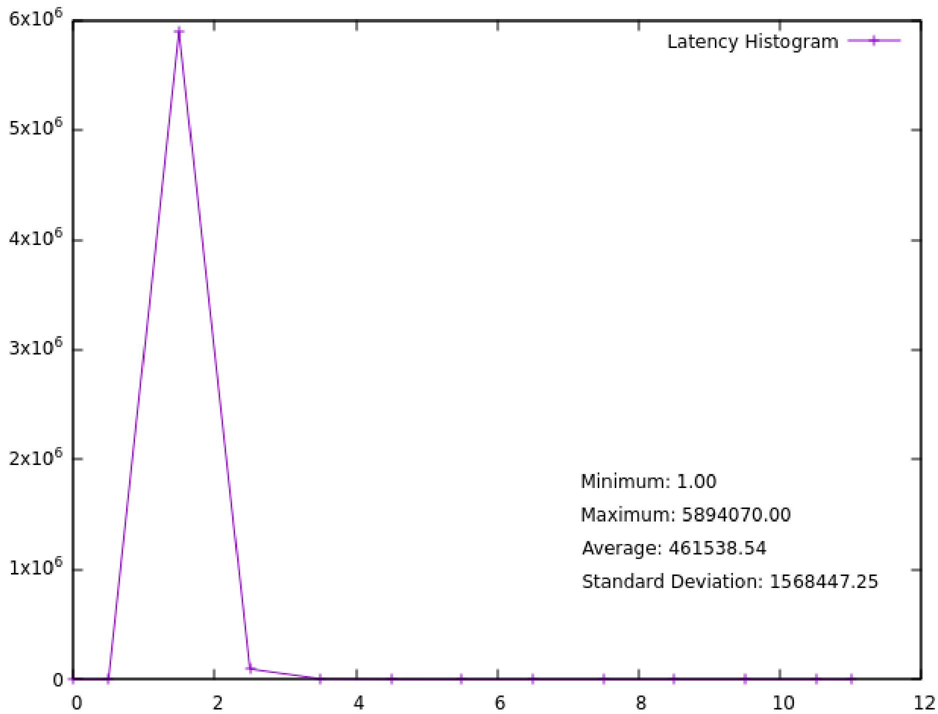

Figure 10: gnuplot latency hardware

Figure 11

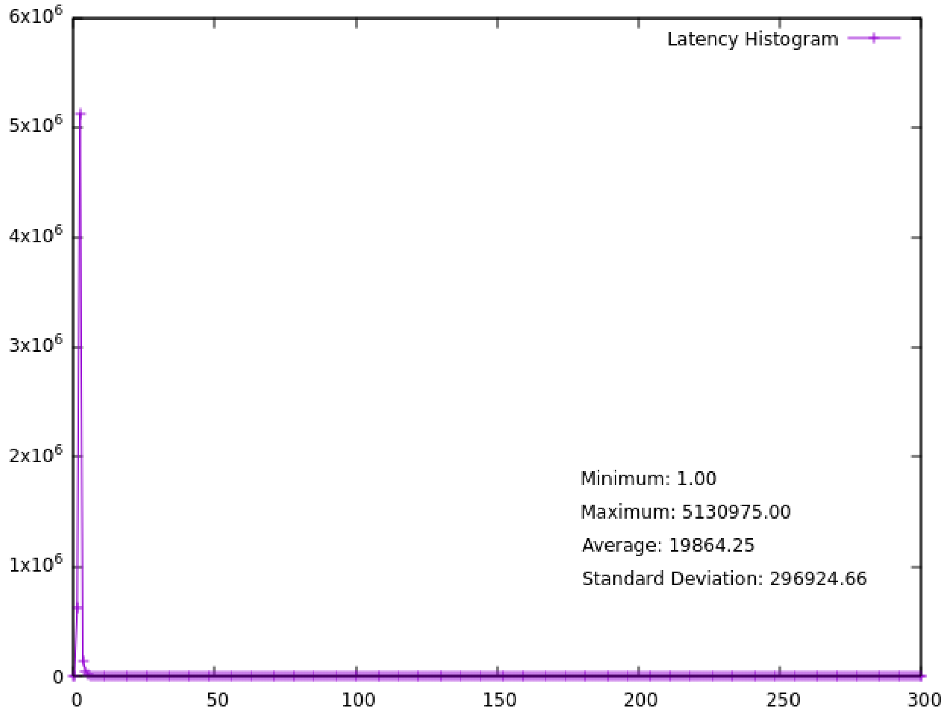

Figure 11: gnuplot latency no taskset

Figure 12

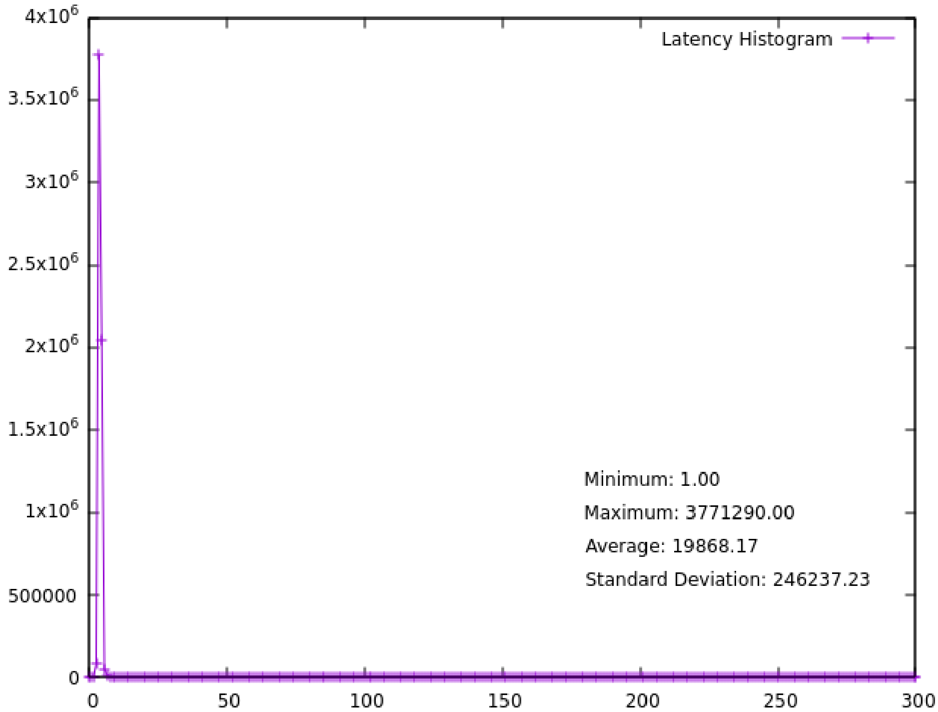

Figure 12: gnuplot latency with taskset

Figure 13

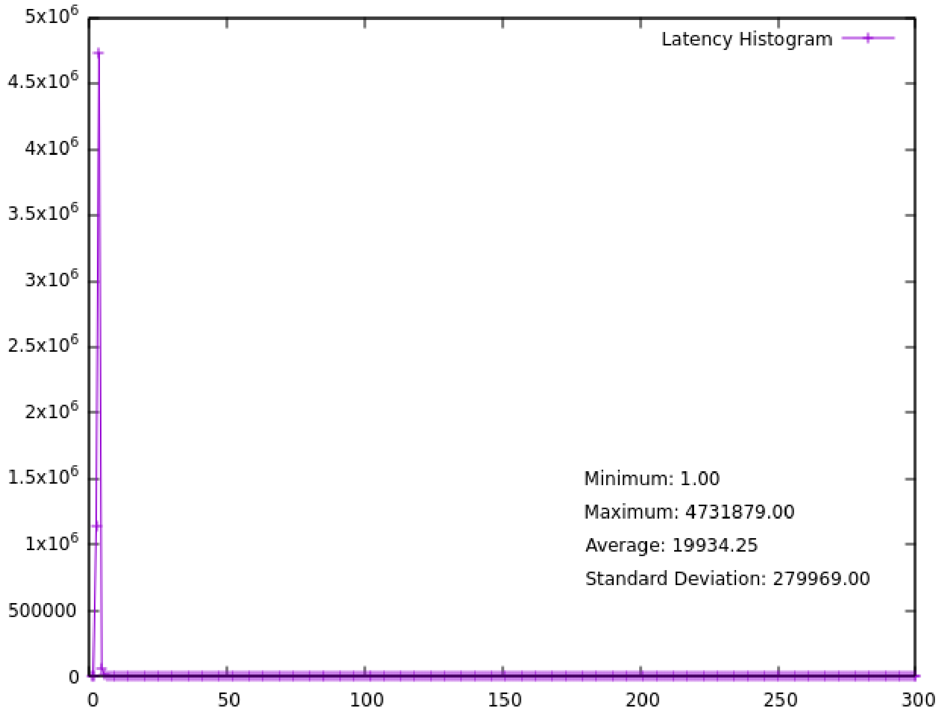

Figure 13: gnuplot latency with taskset

3.4 Latency Comparison

In the initial phase, a comparative latency analysis was conducted between the hardware version and the virtualized version of Salamander 4. For this purpose, the latency tool of the Xenomai test suite was used. The latency was measured under two conditions, idle and CPU-stressed. The goal was to optimize the latency of the virtualisation of Salamander 4 OS to closely match that of the bare metal version.

Vorgehensweise von [7]

Table 5: Kernel and Patches

Kernel	**Patches**
.	.
.	.

After analyzing the inital latency of both versions, Trace-cmd and Kernelshark were used to further inspect the reasons that caused this divergence.

4 Real-Time Latency Reduction

4.1 CPU isolation

Isolating CPUs involves removing all user-space threads and unbound kernel threads since bound kernel threads are tied to specific CPUs and hence cannot be moved. Also, modifying the `proc/irq/IRQ_NUMBER/smp_affinity` property of each Interrupt IRQ_NUMBER in the system is part of this process, as described later in section 4.2. Output 4 shows the user and kernel tasks that run on CPU 19. After the isolation, user tasks other than the QEMU process have been removed from running on this CPU. Only few critical kernel threads that are tied to this CPU still take CPU time.

```
sigma_ibo@sigma-ibo:~$ cat /sys/devices/system/cpu/isolated
19
sigma_ibo@sigma-ibo:~$ ps -e -o pid,psr,comm | awk '$2 == 19'
     92  19 cpuhp/19
     93  19 idle_inject/19
     94  19 migration/19
     95  19 ksoftirqd/19
     97  19 kworker/19:0H-events_highpri
   1025  19 irq/205-iwlwifi:queue_7
  17448  19 kworker/19:1H-kblockd
  17499  19 kworker/19:2-events
  18761  19 kworker/19:3-events
  21401  19 qemu-system-x86
```

Code 4: User and Kernel Tasks

4.2 Interrupt Requests Handling

Once the CPUs were isolated, Interrupt Requests Handling was the next step. Interrupt Requests are used to send a signal to the CPU, prompting it to 'interrupt' its current task and divert its attention to another task. This allows hardware devices to communicate with the CPU through frequent context switches, which can lead to performance degradation, especially in high-performance computing or real-time scenarios. To mitigate this, the IRQs needed to be removed from the isolated CPUs. This was done by manipulating a file in the proc filesystem, namely `/proc/irq/default_smp_affinity`. The value in the `smp_affinity` file is a bitmask in hexadecimal format. Each bit in this mask corresponds to a CPU in the system. The least significant bit (LSB) on the right corresponds to the first CPU (CPU0), and the significance increases towards the left until CPU19. In a system with 20 CPUs, if every CPU was reserved for one IRQ, the value for `smp_affinity` would be FFFFF. The script in code 5 was written to check and log the distribution of Interrupt Requests across each CPU in Salamander 4.

```
#!/bin/bash
# Check if a command-line argument is provided
if [ -z "$1" ]; then
   echo "Please provide a CPU number as a command-line argument."
   exit 1
fi
# Get the CPU number from the command-line argument
CPU=$1
# Initialize an empty array to store the IRQ numbers
IRQs=()
for IRQ in /proc/irq/*; do
   if [ -f "$IRQ/smp_affinity" ]; then
      # Read the current smp_affinity
      AFFINITY=$(cat "$IRQ/smp_affinity")
      # Check if the bit for the current CPU is set
      if (( (0x$AFFINITY & (1 << CPU)) != 0 )); then
         # Add the IRQ number to the array
         IRQs+=("${IRQ#/proc/irq/}")
      fi
   fi
done
# Sort the array
IFS=$'\n' sorted=($(sort -n <<<"${IRQs[*]}"))
# Print the CPU number
echo "CPU $CPU IRQ affinity:"
# Print the sorted IRQ numbers on separate lines
for irq in "${sorted[@]}"; do
   echo "$irq"
done
```

Code 5: Check distribution of Interrupt Requests across each CPU

Output 6 shows the output of the script above for CPU 19.

```
sigma_ibo@sigma-ibo:~$ ./check_smp_affinity.sh 19
CPU 19 IRQ affinity:
0
2
3
4
5
6
7
10
11
13
15
131
172
188
189
195
```

Code 6: Output of smp_affinity for CPU 19

By changing the values in
, I could effectively control the assignment of IRQs, ensuring they would not be handled by the isolated CPUs. This manipulation was done carefully, considering the implications and ensuring the stability and integrity of the system.

Through this approach, I was able to create a more streamlined and efficient environment for the execution of specific tasks on the isolated CPUs. By reducing the interruptions caused by IRQs, the isolated CPUs were able to focus more on their assigned tasks, potentially leading to improved performance and more predictable execution times.

4.3 Kernel tuning

4.4 KVM exit reasons

4.4.1 APIC_WRITE

The Advanced Programmable Interrupt Controller (APIC) is responsible for the distribution of interrupts in x86 and Itanium-based computer systems. It consists of two main components: the I/O APIC and the local APICs. The I/O APIC receives interrupt requests from devices and distributes them as messages to the Local APICs. The Local APICs then forward the highest-priority interrupt to the CPU core. The APIC offers many advantages, including more inputs for interrupts, a flexible configuration, definable priorities and support for message-signalled interrupts.

[8]

APIC virtualization

Newer Intel processors offer hardware virtualization of the Advanced Programmable Interrupt Controller (APICv). APICv improves virtualized AMD64 and Intel 64 guest performance by allowing the guest to directly access the APIC, dramatically cutting down interrupt latencies and the number of virtual machine exits caused by the APIC. This feature is used by default in newer Intel processors and improves I/O performance.

4.4.2 HLT

Description

Occurence Reduction

4.4.3 EPT_MISCONFIG

Description

Occurence Reduction

4.4.4 PREEMPTION_TIMER

Description

Occurence Reduction

4.4.5 EXTERNAL_INTERRUPT

Description

Occurence Reduction

4.4.6 IO_INSTRUCTION

Description

Occurence Reduction

4.4.7 EOI_INDUCED

Description

Occurence Reduction

4.4.8 EPT_VIOLATION

Description

Occurence Reduction

4.4.9 PAUSE_INSTRUCTION

Description

Occurence Reduction

4.4.10 CPUID

Description

Occurence Reduction

4.4.11 MSR_READ

Description

Occurence Reduction

5 Results

6 Discussion

7 Summary and Outlook

Bibliography

[1] pixelart. *SIGMATEK - Komplette Automatisierungssysteme*. URL: https://www.sigmatek-automation.com/de/ (visited on 03/27/2024).

[2] *Trace-Cmd*. URL: https://trace-cmd.org/ (visited on 03/25/2024).

[3] *KernelShark*. URL: https://kernelshark.org/ (visited on 03/25/2024).

[4] *Xenomai :: Xenomai*. URL: https://xenomai.org/ (visited on 03/21/2024).

[5] *Welcome to the Yocto Project Documentation — The Yocto Project ® 4.3.999 Documentation*. URL: https://docs.yoctoproject.org/ (visited on 03/27/2024).

[6] *QEMU*. URL: https://www.qemu.org/ (visited on 03/27/2024).

[7] Chan-Hsiang Lin and Che-Kang Wu. “Performance Evaluation of Xenomai 3”. In: ().

[8] Petro Lutsyk, Jonas Oberhauser, and Wolfgang J. Paul. *A Pipelined Multi-Core Machine with Operating System Support: Hardware Implementation and Correctness Proof*. Lecture Notes in Computer Science Theoretical Computer Science and General Issues 9999. Cham: Springer, 2020. 628 pp. ISBN: 978-3-030-43242-3.

List of Figures

List of Tables

List of Code

List of Abbreviations

CPU Central Processing Unit

QEMU Quick Emulator

IRQ Interrupt Request

A Anhang A

B Anhang B