

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, May 29, 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, May 29, 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

Sections 4.1 and 4.2 demonstrate the initial real-time latency values gathered for bare metal and virtualisation.

Keywords: Echtzeit, Virtualisierung, Xenomai, VARAN

Contents

1	Introduction	1
1.1	Application Context	2
1.2	State of the art	2
1.3	Problem and task definition	2
1.4	Objective	2
2	Methodology	3
3	Salamander 4	5
3.1	Structure	5
3.2	Memory Management	6
3.3	Xenomai	7
4	Initial Real-Time Latency	8
4.1	Salamander 4 Bare Metal	8
4.2	Salamander 4 Virtualisation	10
5	Real-Time Performance Tuning	14
5.1	BIOS Configurations	14
5.2	Kernel Configurations	14
5.3	Host OS Configurations	15
5.3.1	CPU isolation	15
5.3.2	Interrupt Requests Handling	17
5.3.3	Disable simultaneous multithreading	19
5.3.4	Disable dynamic frequency scaling	19
5.3.5	Disable RT throttling	19
5.3.6	no unexpected RT processes are running on your system	19
5.3.7	IRQ affinity	19
5.3.8	RCU CPU offloading	19
5.3.9	Suppress rcu cpu stall	19
5.3.10	Maybe?	19
5.3.11	Start QEMU normally and give all QEMU threads rt-priority	19
5.3.12	Kill all running user processes	19
5.3.13	Set CPU Affinity for systemd services	19
5.3.14	Cache Isolation for CPU and GPU	19

5.3.15 Set CPU Affinity of IRQ thread to CPU 0	19
5.3.16 Set Device Driver Work Queue to CPU 0	19
5.3.17 Disable Machine Check	19
5.3.18 Stop Certain Services	19
5.4 QEMU-KVM Configurations	19
5.5 Guest OS Configurations	19
6 KVM exit reasons	21
6.1 APIC_WRITE	21
6.2 HLT	21
6.3 EPT_MISCONFIG	21
6.4 PREEMPTION_TIMER	21
6.5 EXTERNAL_INTERRUPT	21
6.6 IO_INSTRUCTION	21
6.7 EOI_INDUCED	21
6.8 EPT_VIOLATION	21
6.9 PAUSE_INSTRUCTION	21
6.10 CPUID	21
6.11 MSR_READ	21
7 Real-Time Robotic Application	22
7.1 VARAN	22
7.2 Robotic Application	22
8 Results	23
9 Discussion	24
10 Summary and Outlook	25
Bibliography	26
List of Figures	29
List of Tables	30
List of Code	31
List of Abbreviations	32
A Anhang A	33
B Anhang B	34

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, namely Salamander 4, to be run on their self-manufactured CPUs. Salamander 4 system employs hard real-time with Xenomai 3 and requires a worst latency value between 20 and 50 μ s. The goal is to virtualize Salamander 4 and approach the performance of bare metal. Salamander 4 is built with Yocto and virtualized through QEMU/KVM. 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 3 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].

The system configuration is shown in Table 1

Table 1: System configuration

CPU	13th Gen Intel(R) Core(TM) i7-13800H
Memory	2x 16GB SO-DIMM DDR5-5600 MT/s, 32GB
GPU	NVIDIA RTX A500 Laptop GPU
BIOS	Dell Version 1.12.0
OS	Ubuntu 22.04.4 LTS

Figure 1 is the output of the `lstopo` command and visualizes the hardware nodes of the system, including CPU cores, caches, memory, and I/O devices.

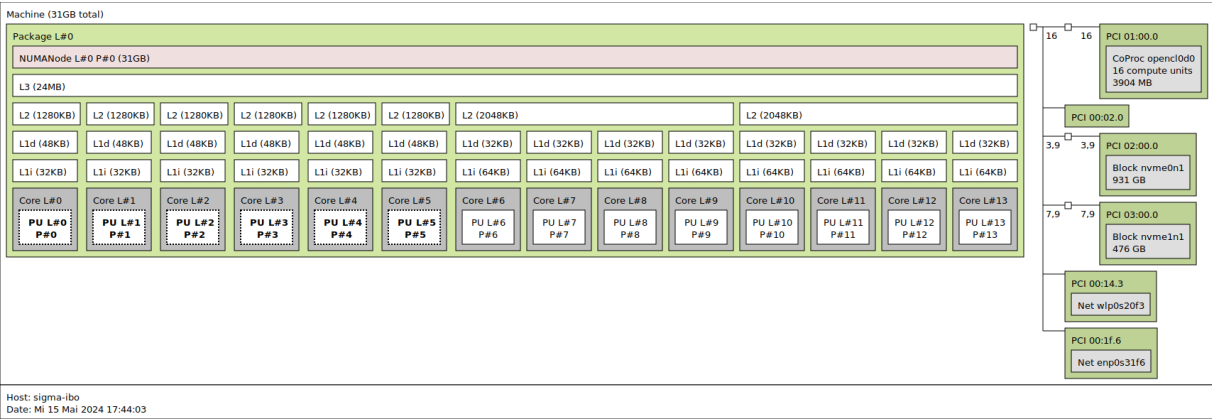


Figure 1: Hardware topology

3 Salamander 4

This chapter briefly describes the Salamander 4 operating system by SIGMATEK.

3.1 Structure

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.

```
1 root@sigmatek-core2:~# uname -a
2 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

Salamander 4 is powered by SIGMATEK's CP 841 [5] and is comprised of the following software modules:

- **Operating system:** The operating system in a LASAL CPU manages the hardware and software resources of the system. It is provided in a completely PC-compatible manner, working with a standard PC BIOS.
- **Loader:** The loader is a part of the operating system that is responsible for loading programs from executables into memory, preparing them for execution and then executing them.
- **Hardware classes:** Hardware classes in LASAL represent the different types of hardware components that can be controlled by the LASAL CPU. They provide a way to organize and manage the hardware components in a modular and reusable manner. The graphical hardware editor in LASAL allows for a true-to-detail simulation of the actual hardware.
- **Application:** Applications are developed using LASAL CLASS 2 [6], a solution for automation tasks that supports object-oriented programming and design in compliance with IEC 61131-3.

The interfaces between the individual modules are indicated by an arrow in Figure 2.



Figure 2: Structure of Salamander 4 CPU

3.2 Memory Management

For the sake of completeness, Figure 3 displays the memory management of Salamander 4. LRT stands for Lasal Runtime and creates an execution environment where applications developed using the LASAL Class 2 can run, providing defined real-time functions, data types, and other constructs tailored for real-time programming.



Figure 3: Memory Management

3.3 Xenomai

Xenomai 3 is a real-time framework that offers two paths to real-time performance. The first approach supplements the Linux kernel with a compact real-time core dubbed Cobalt, demonstrated in Figure 4. Cobalt runs side-by-side with Linux, but it handles all time-critical activities like interrupt processing and real-time thread scheduling with higher priority than the regular kernel activities.

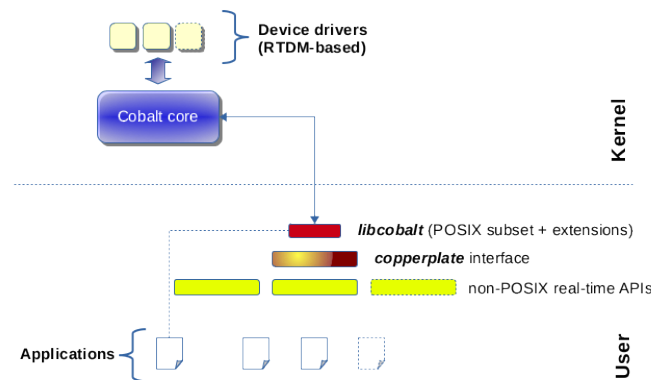


Figure 4: Xenomai Cobalt interfaces

The second approach, called Mercury and shown in Figure 5, relies on the real-time capabilities already present in the native Linux kernel. Often, applications require the PREEMPT-RT extension to augment the mainline kernel's real-time responsiveness and minimize jitter, but this isn't mandatory and depends on the application's specific requirements for responsiveness and tolerance for occasional deadline misses.

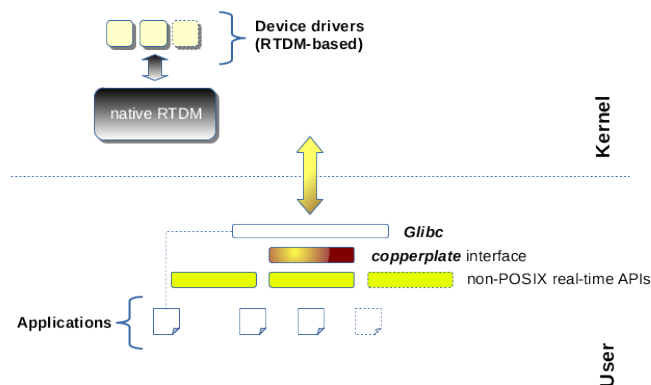


Figure 5: Xenomai Mercury interfaces

Salamander 4 uses the Cobalt real-time core with the Dovetail extension, which allows the kernel to handle real-time tasks with low latency.

4 Initial Real-Time Latency

As a starting point, initial latency values of both the bare metal and virtualization versions were measured with the latency tool of the xenomai tool suite. Salamander 4 Bare Metal refers to the proprietary hardware of SIGMATEK used to employ the custom operating system. Salamander 4 virtualisation refers to a virtual version of the Salamander 4 hardware platform, achieved through QEMU/KVM. Sections 4.1 and 4.2 specify the details of the measurements for both versions. In the further course, the aim was to bring the latency values of the virtualization closer to those of the hardware.

4.1 Salamander 4 Bare Metal

The output of the command `uname -a` for Salamander 4 bare metal is shown in code 2.

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

Code 2: Salamander 4 bare metal system information

As a reference point, the latency program was executed on Salamander 4 bare metal for a duration of 10 minutes. The complete command used was `latency -h -s -T 600`. Figure 6 shows the gathered latency values in microseconds.

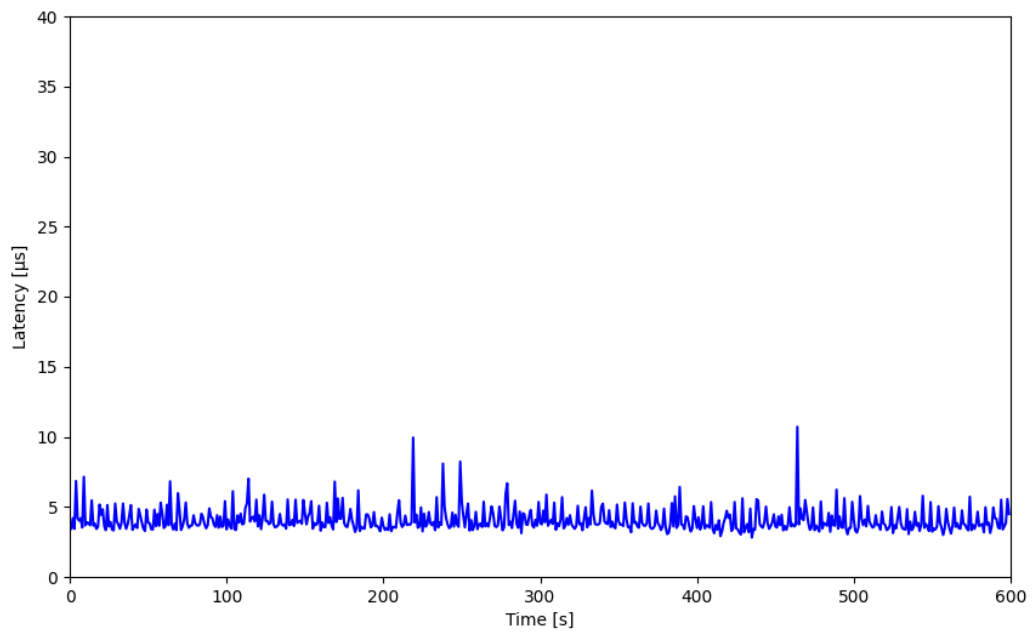


Figure 6: Latency hardware

Figure 9 depicts the variation in latency over the course of said time.

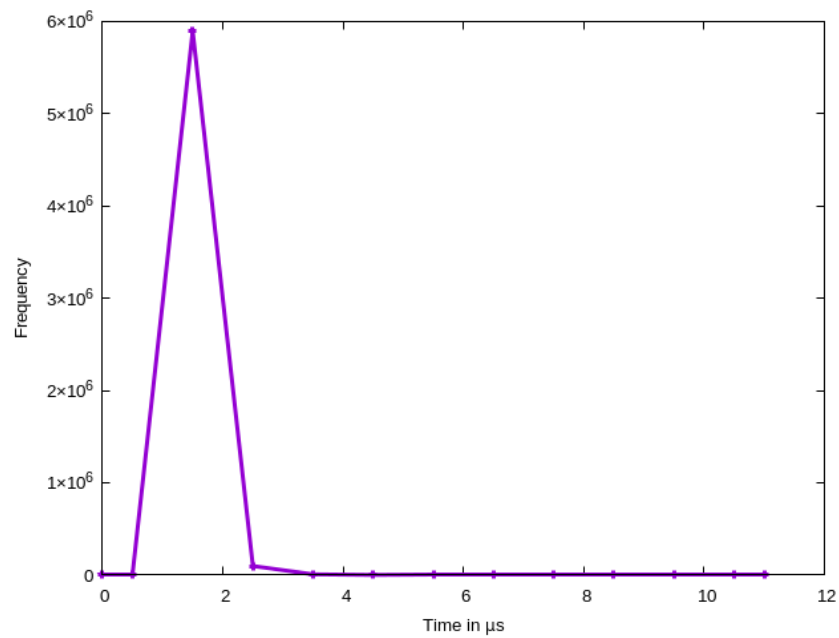


Figure 7: gnuplot latency hardware

The statistics obtained from this measurement are provided in Table 2. It gives an overview of the average, maximum, minimum latency, and the standard deviation of the latency values in microseconds.

Table 2: Latency Statistics in microseconds

Statistic	Value (in μ s)
Average Latency	4.06
Maximum Latency	10.71
Minimum Latency	2.82
Standard Deviation	0.85

4.2 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 developed using Yocto, an open source project that allows customised Linux distributions to be created for embedded systems [7]. The virtualisation runs in a QEMU environment, which is an open source tool for hardware virtualisation [8]. With the help of the script depicted in code 4, 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 3.

```

1  sigma_ibo@localhost:~/Desktop/salamander-image$ ls -l
2  bzImage
3  drive-c
4  ovmf.code.qcow2
5  qemu_def.sh
6  salamander-image-sigmatek-core2.ext4
7  stek-drive-c-image-sigmatek-core2.tar.gz
8  vmlinux

```

Code 3: Contents of QEMU folder for Salamander 4

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 Salamander 4 OS.

- **stek-drive-c-image-sigmathek-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-sigmathek-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.

The initial QEMU script after the custom Yocto build and the starting point for this work is shown in Code 4. This script is used to start QEMU with correct parameters to boot Salamander 4 OS. It will be adjusted in chapter 5 in order to accompany real-time performance tunings.

```

1  #!/bin/sh
2
3  if [ ! -d drive-c/ ]; then
4      echo "Filling drive-c/"
5      mkdir drive-c/
6      tar -C drive-c/ -xf stek-drive-c-image-sigmathek-core2.tar.gz
7  fi
8
9  exec qemu-system-x86_64 -M pc,accel=kvm -kernel ./bzImage \
10 -m 2048 -drive
      file=salamander-image-sigmathek-core2.ext4,format=raw,media=disk \
11 -append "console=ttyS0 console=tty1 root=/dev/sda rw panic=1
      sigmatek_lrt.QEMU=1 ip=dhcp rootfstype=ext4 schedstats=enable" \
12 -net nic,model=e1000,netdev=e1000 -netdev bridge,id=e1000,br=nm-bridge \
13 -fsdev local,security_model=none,id=fsdev0,path=drive-c -device
      virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=/mnt/drive-C \
14 -device vhost-vsock-pci,guest-cid=3,id=vsock0 \
15 -drive if=pflash,format=qcow2,file=ovmf.code.qcow2 \
16 -no-reboot -nographic

```

Code 4: QEMU script for starting Salamander 4 virtualisation

The script is run on a generic Ubuntu 22.04.4 system. The kernel version and other details are presented in Code 5, using the `uname -a` command.

```

1 root@sigmatek-core2:~# uname -a
2 Linux sigma-ibo 6.5.0-35-generic #35~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue
   May  7 09:00:52 UTC 2 x86_64 x86_64 x86_64 GNU/Linux

```

Code 5: Ubuntu 22.04.4 system information

Measuring the latency of the Salamander 4 virtualization with the default QEMU script in Code 4 and no further adjustments for 10 minutes, the following latency values in Figure 8 were collected.

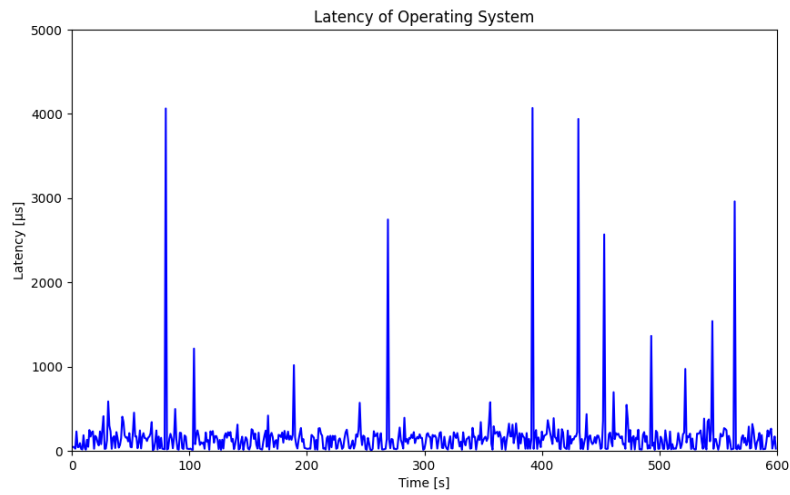


Figure 8: Latency no taskset

Figure ?? depicts the variation in latency over the course of said time.

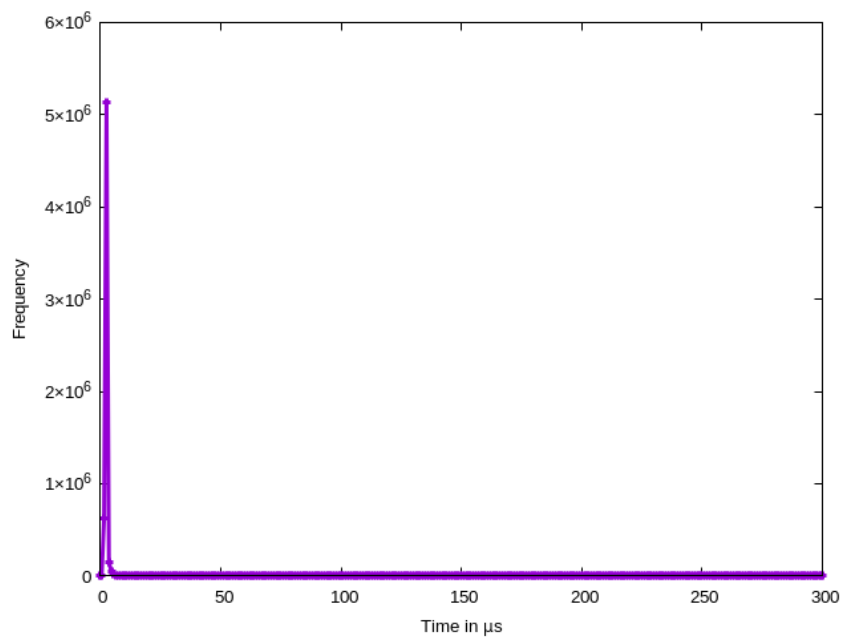


Figure 9: gnuplot latency hardware

The statistics obtained from this measurement are provided in Table 3. It gives an overview of the average, maximum, minimum latency, and the standard deviation of the latency values in microseconds.

Table 3: Latency Statistics default in microseconds

Statistic	Value (in μs)
Average Latency	174.5us
Maximum Latency	4070.02us
Minimum Latency	6.96us
Standard Deviation	359.27us

5 Real-Time Performance Tuning

5.1 BIOS Configurations

Table 4: BIOS Configurations

Option	Status
Hyper Threading	Disabled
Intel SpeedStep®	Disabled
Intel® Speed Shift Technology	Disabled
C States	Disabled
VT-d	Enabled

5.2 Kernel Configurations

The kernel command-line parameters [6] are shown in Code 6 below. Some of them are essential for real-time performance.

```
1 GRUB_CMDLINE_LINUX="isolcpus=4 rcu_nocbs=4 nohz_full=4
    default_hugepagesz=1G hugepagesz=1G hugepages=8 intel_iommu=on
    rdt=l3cat nmi_watchdog=0 idle=poll clocksource=tsc tsc=reliable
    noht audit=0 skew_tick=1 intel_pstate=disable intel.max_cstate=0
    intel_idle.max_cstate=0 processor.max_cstate=0
    processor_idle.max_cstate=0 nosoftlockup nohz=on no_timer_check
    nospectre_v2 spectre_v2_user=off kvm.kvmclock_periodic_sync=N
    kvm_intel.ple_gap=0 irqaffinity=0"
```

Code 6: Kernel Configuration

In the following, these settings along with their impact on system latency are briefly described.

- **isolcpus=4**: This isolates CPU 4 from the general scheduler, meaning no process will be scheduled to run on this CPU unless it is explicitly assigned.
- **rcu_nocbs=4**: Moves the RCU (Read-Copy-Update) callback handling off of CPU 4.
- **nohz_full=4**: Makes CPU 4 tickless, reducing the timer interrupts and thus can improve performance for specific applications.

- **default_hugepagesz=1G hugepagesz=1G hugepages=8**: Sets the default huge page size to 1GB, and reserves 8 huge pages at boot.
- **intel_iommu=on**: Enables Intel's IOMMU (Input/Output Memory Management Unit) for hardware that supports it.
- **rdt=l3cat**: Enables the L3 Cache Allocation Technology.
- **nmi_watchdog=0**: Disables the NMI watchdog, which can free up a bit of system resources.
- **idle=poll**: Changes the idle loop of the CPU to actively poll.
- **clocksource=tsc tsc=reliable**: Sets the clocksource to TSC (Time Stamp Counter) and marks it as reliable.
- **noht**: Disables Hyper-Threading.
- **audit=0**: Disables the Linux audit system.
- **skew_tick=1**: Enables a mode to reduce timer interrupt overhead.
- **intel_pstate=disable**: Disables the intel_pstate driver.
- **intel.max_cstate=0 intel_idle.max_cstate=0 processor.max_cstate=0 processor_idle.max_cstate=0**: Disables deeper C-states, which can help reduce latency.
- **nosoftlockup**: Disables the soft lockup detector.
- **nohz=on**: Enables tickless operation system-wide.
- **no_timer_check**: Disables the check for broken timer interrupt sources.
- **nospectre_v2 spectre_v2_user=off**: Disables mitigations for the Spectre v2 vulnerability.
- **kvm.kvmclock_periodic_sync=N kvm_intel.ple_gap=0**: These are KVM (Kernel-based Virtual Machine) related parameters.
- **irqaffinity=0**: Sets the default IRQ affinity.

5.3 Host OS Configurations

5.3.1 CPU isolation

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 [9].

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 5.3.2. Output 7 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.

```
1      sigma_ibo@sigma-ibo:~$ cat /sys/devices/system/cpu/isolated
2      19
3      sigma_ibo@sigma-ibo:~$ ps -e -o pid,psr,comm | awk '$2 == 19'
4          92  19  cpuhp/19
5          93  19  idle_inject/19
6          94  19  migration/19
7          95  19  ksoftirqd/19
8          97  19  kworker/19:0H-events_highpri
9         1025  19  irq/205-iwlwifi:queue_7
10        17448  19  kworker/19:1H-kblockd
11        17499  19  kworker/19:2-events
12        18761  19  kworker/19:3-events
13        21401  19  qemu-system-x86
```

Code 7: User and Kernel Tasks

5.3.2 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/<IRQ>/smp_affinity`. The value in the `smp_affinity` file is a bit-mask 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 8 was written to check and log the distribution of Interrupt Requests across each CPU in Salamander 4.

```
1      #!/bin/bash
2      # Check if a command-line argument is provided
3      if [ -z "$1" ]; then
4          echo "Please provide a CPU number as a command-line argument."
5          exit 1
6      fi
7      # Get the CPU number from the command-line argument
8      CPU=$1
9      # Initialize an empty array to store the IRQ numbers
10     IRQs=()
11     for IRQ in /proc/irq/*; do
12         if [ -f "$IRQ/smp_affinity" ]; then
13             # Read the current smp_affinity
14             AFFINITY=$(cat "$IRQ/smp_affinity")
15             # Check if the bit for the current CPU is set
16             if (( (0xAFFINITY & (1 << CPU)) != 0 )); then
17                 # Add the IRQ number to the array
18                 IRQs+=("$IRQ#/proc/irq/")
19             fi
20         fi
21     done
22     # Sort the array
23     IFS=$'\n' sorted=$(sort -n <<<"${IRQs[*]}")
24     # Print the CPU number
25     echo "CPU $CPU IRQ affinity:"
26     # Print the sorted IRQ numbers on separate lines
27     for irq in "${sorted[@]}"; do
28         echo "$irq"
29     done
```

Code 8: Check distribution of Interrupt Requests across each CPU

Output 9 shows the output of the script above for CPU 19.

```
1  sigma_ibo@sigma-ibo:~$ ./check_smp_affinity.sh 19
2  CPU 19 IRQ affinity:
3  0
4  2
5  3
6  4
7  5
8  6
9  7
10 10
11 11
12 13
13 15
14 131
15 172
16 188
17 189
18 195
```

Code 9: Output of smp_affinity for CPU 19

By changing the values of the `smp_affinity` files of the respective IRQs, the assignment of IRQs was controlled so that they would not be handled by the isolated CPU. This reduced the interruptions caused by IRQs and the isolated CPUs were able to focus more on their assigned tasks.

- 5.3.3 Disable simultaneous multithreading
- 5.3.4 Disable dynamic frequency scaling
- 5.3.5 Disable RT throttling
- 5.3.6 no unexpected RT processes are running on your system
- 5.3.7 IRQ affinity
- 5.3.8 RCU CPU offloading
- 5.3.9 Suppress rcu cpu stall
- 5.3.10 Maybe?
- 5.3.11 Start QEMU normally and give all QEMU threads rt-priority
- 5.3.12 Kill all running user processes
- 5.3.13 Set CPU Affinity for systemd services
- 5.3.14 Cache Isolation for CPU and GPU
- 5.3.15 Set CPU Affinity of IRQ thread to CPU 0
- 5.3.16 Set Device Driver Work Queue to CPU 0
- 5.3.17 Disable Machine Check
- 5.3.18 Stop Certain Services

5.4 QEMU-KVM Configurations

5.5 Guest OS Configurations

```

17  #!/bin/sh
18
19  if [ ! -d drive-c/ ]; then
20      echo "Filling drive-c/"
21      mkdir drive-c/
22      tar -C drive-c/ -xf stek-drive-c-image-sigmatek-core2.tar.gz
23  fi
24
25  exec qemu-system-x86_64 -M pc,accel=kvm -kernel ./bzImage \
26  -m 2048 -drive
27      file=salamander-image-sigmatek-core2.ext4,format=raw,media=disk \
28  -append "console=ttyS0 console=tty1 root=/dev/sda rw panic=1
29      sigmatek_lrt.QEMU=1 ip=dhcp rootfstype=ext4 schedstats=enable" \
30  -net nic,model=e1000,netdev=e1000 -netdev bridge,id=e1000,br=nm-bridge \
31  -fsdev local,security_model=none,id=fsdev0,path=drive-c -device
32      virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=/mnt/drive-C \
33  -device vhost-vsock-pci,guest-cid=3,id=vsock0 \
34  -drive if=pflash,format=qcow2,file=ovmf.code.qcow2 \
35  -no-reboot -nographic

```

Code 10: QEMU script for starting Salamander 4 virtualisation

6 KVM exit reasons

6.1 APIC_WRITE

6.2 HLT

6.3 EPT_MISCONFIG

6.4 PREEMPTION_TIMER

6.5 EXTERNAL_INTERRUPT

6.6 IO_INSTRUCTION

6.7 EOI_INDUCED

6.8 EPT_VIOLATION

6.9 PAUSE_INSTRUCTION

6.10 CPUID

6.11 MSR_READ

7 Real-Time Robotic Application

7.1 VARAN

7.2 Robotic Application

8 Results

[1] [2] [3] [4] [5] [6] [7] [8] [10] [9] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24]
[25] [26] [27] [28] [29] [30]

9 Discussion

10 Summary and Outlook

Bibliography

- [1] pixelart. *SIGMATEK - Komplette Automatisierungssysteme*. <https://www.sigmatek-automation.com/de/>. (Visited on 03/27/2024).
- [2] *Trace-Cmd*. <https://trace-cmd.org/>. (Visited on 03/25/2024).
- [3] *KernelShark*. <https://kernelshark.org/>. (Visited on 03/25/2024).
- [4] *Xenomai :: Xenomai*. <https://xenomai.org/>. (Visited on 03/21/2024).
- [5] *CPU-Einheiten* - *SIGMATEK*. <https://www.sigmatek-automation.com/de/produkte/steuerungssysteme/cpu-einheiten/cp-841/>. (Visited on 05/27/2024).
- [6] *Engineering Tool LASAL* - *SIGMATEK*. <https://www.sigmatek-automation.com/de/produkte/engineering-tool-lasal/lasal-class/>. (Visited on 05/27/2024).
- [7] *Welcome to the Yocto Project Documentation — The Yocto Project @ 4.3.999 Documentation*. <https://docs.yoctoproject.org/>. (Visited on 03/27/2024).
- [8] *QEMU*. <https://www.qemu.org/>. (Visited on 03/27/2024).
- [9] Ruhui Ma et al. "Performance Tuning Towards a KVM-based Embedded Real-Time Virtualization System". In: ().
- [10] Chan-Hsiang Lin and Che-Kang Wu. "Performance Evaluation of Xenomai 3". In: ().
- [11] S. Brosky and S. Rotolo. "Shielded Processors: Guaranteeing Sub-Millisecond Response in Standard Linux". In: *Proceedings International Parallel and Distributed Processing Symposium*. Nice, France: IEEE Comput. Soc, 2003, p. 9. ISBN: 978-0-7695-1926-5. DOI: [10.1109/IPDPS.2003.1213237](https://doi.org/10.1109/IPDPS.2003.1213237). (Visited on 04/18/2024).
- [12] Marcello Cinque et al. "Virtualizing Mixed-Criticality Systems: A Survey on Industrial Trends and Issues". In: *Future Generation Computer Systems* 129 (Apr. 2022), pp. 315–330. ISSN: 0167739X. DOI: [10.1016/j.future.2021.12.002](https://doi.org/10.1016/j.future.2021.12.002). arXiv: [2112.06875](https://arxiv.org/abs/2112.06875) [cs]. (Visited on 03/25/2024).
- [13] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. "Challenges in Real-Time Virtualization and Predictable Cloud Computing". In: *Journal of Systems Architecture* 60.9 (Oct. 2014), pp. 726–740. ISSN: 13837621. DOI: [10.1016/j.sysarc.2014.07.004](https://doi.org/10.1016/j.sysarc.2014.07.004). (Visited on 03/25/2024).

- [14] Zonghua Gu and Qingling Zhao. “A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization”. In: *Journal of Software Engineering and Applications* 05.04 (2012), pp. 277–290. ISSN: 1945-3116, 1945-3124. DOI: [10.4236/jsea.2012.54033](https://doi.org/10.4236/jsea.2012.54033). (Visited on 03/25/2024).
- [15] “Hard Real Time Linux* Using Xenomai* on Intel® Multi-Core Processors”. In: ().
- [16] Diogenes Javier Perez et al. “How Real (Time) Are Virtual PLCs?” In: *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. Stuttgart, Germany: IEEE, Sept. 2022, pp. 1–8. ISBN: 978-1-66549-996-5. DOI: [10.1109/ETFA52439.2022.9921545](https://doi.org/10.1109/ETFA52439.2022.9921545). (Visited on 03/25/2024).
- [17] Veronika Kirova et al. “Impact of Modern Virtualization Methods on Timing Precision and Performance of High-Speed Applications”. In: *Future Internet* 11.8 (Aug. 2019), p. 179. ISSN: 1999-5903. DOI: [10.3390/fi11080179](https://doi.org/10.3390/fi11080179). (Visited on 03/25/2024).
- [18] Jan Kiszka. “Towards Linux as a Real-Time Hypervisor”. In: ().
- [19] 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. ISBN: 978-3-030-43242-3.
- [20] HayfaaSubhi Malallah et al. “A Comprehensive Study of Kernel (Issues and Concepts) in Different Operating Systems”. In: *Asian Journal of Research in Computer Science* (May 2021), pp. 16–31. ISSN: 2581-8260. DOI: [10.9734/ajrcos/2021/v8i330201](https://doi.org/10.9734/ajrcos/2021/v8i330201). (Visited on 03/25/2024).
- [21] Alejandro Masrur et al. “VM-Based Real-Time Services for Automotive Control Applications”. In: *2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*. Macau, China: IEEE, Aug. 2010, pp. 218–223. ISBN: 978-1-4244-8480-5. DOI: [10.1109/RTCSA.2010.38](https://doi.org/10.1109/RTCSA.2010.38). (Visited on 03/25/2024).
- [22] Paul E McKenney. “‘Real Time’ vs. ‘Real Fast’: How to Choose?” In: ().
- [23] Éric Piel et al. “Asymmetric Scheduling and Load Balancing for Real-Time on Linux SMP”. In: *Parallel Processing and Applied Mathematics*. Ed. by David Hutchison et al. Vol. 3911. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 896–903. ISBN: 978-3-540-34141-3 978-3-540-34142-0. DOI: [10.1007/11752578_108](https://doi.org/10.1007/11752578_108). (Visited on 05/06/2024).
- [24] Rui Queiroz, Tiago Cruz, and Paulo Simões. “Testing the Limits of General-Purpose Hypervisors for Real-Time Control Systems”. In: *Microprocessors and Microsystems* 99 (June 2023), p. 104848. ISSN: 01419331. DOI: [10.1016/j.micpro.2023.104848](https://doi.org/10.1016/j.micpro.2023.104848). (Visited on 03/25/2024).
- [25] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. “The Real-Time Linux Kernel: A Survey on PREEMPT_RT”. In: *ACM Computing Surveys* 52.1 (Jan. 2020), pp. 1–36. ISSN: 0360-0300, 1557-7341. DOI: [10.1145/3297714](https://doi.org/10.1145/3297714). (Visited on 03/25/2024).

- [26] Hobin Yoon, Jungmoo Song, and Jamee Lee. "Real-Time Performance Analysis in Linux-Based Robotic Systems". In: ().
- [27] George K. Adam, Nikos Petrellis, and Lambros T. Doulos. "Performance Assessment of Linux Kernels with PREEMPT_RT on ARM-Based Embedded Devices". In: *Electronics* 10.11 (June 2021), p. 1331. ISSN: 2079-9292. DOI: [10.3390/electronics10111331](https://doi.org/10.3390/electronics10111331). (Visited on 05/08/2024).
- [28] George K. Adam. "Real-Time Performance and Response Latency Measurements of Linux Kernels on Single-Board Computers". In: *Computers* 10.5 (May 2021), p. 64. ISSN: 2073-431X. DOI: [10.3390/computers10050064](https://doi.org/10.3390/computers10050064). (Visited on 05/16/2024).
- [29] Daniel Bristot de Oliveira et al. "Demystifying the Real-Time Linux Scheduling Latency". In: ().
- [30] "Real-Time Performance Tuning Best Practice Guidelines for KVM-Based Virtual Machines". In: (2022).

List of Figures

Figure 1 Hardware topology	4
Figure 2 Structure of Salamander 4 CPU	6
Figure 3 Memory Management	6
Figure 4 Xenomai Cobalt interfaces	7
Figure 5 Xenomai Mercury interfaces	7
Figure 6 Latency hardware	9
Figure 7 gnuplot latency hardware	9
Figure 8 Latency no taskset	12
Figure 9 gnuplot latency hardware	12

List of Tables

Table 1	System configuration	4
Table 2	Latency Statistics in microseconds	10
Table 3	Latency Statistics default in microseconds	13
Table 4	BIOS Configurations	14

List of Code

Code 1	System information	5
Code 2	Salamander 4 bare metal system information	8
Code 3	Contents of QEMU folder for Salamander 4	10
Code 4	QEMU script for starting Salamander 4 virtualisation	11
Code 5	Ubuntu 22.04.4 system information	11
Code 6	Kernel Configuration	14
Code 7	User and Kernel Tasks	16
Code 8	Check distribution of Interrupt Requests across each CPU	17
Code 9	Output of smp_affinity for CPU 19	18

List of Abbreviations

CPU Central Processing Unit

QEMU Quick Emulator

IRQ Interrupt Request

A Anhang A

B Anhang B