

# **Linux und Echtzeit**

## **< 10 usec auch auf der ARM-Architektur?**

Jan Altenberg / Heinz Egger, linutronix GmbH

**Linux ist aufgrund der hohen Anzahl unterstützter CPU Architekturen, der nahezu unendlichen Anzahl von Treibern und nicht zuletzt der guten Portierbarkeit und Skalierbarkeit eines der leistungsfähigsten Embedded Betriebssysteme unserer Zeit! Auch Systeme mit harten Echtzeitanforderungen können mit Linux einfach umgesetzt werden. Hierfür gibt es unterschiedliche Varianten und Ansätze. Doch welcher Ansatz ist der richtige? Und welche Latenzzeiten können damit erreicht werden? Dieser Artikel stellt unterschiedliche Technologien vor, mit denen harte Echtzeitfähigkeit unter Linux erreicht werden kann. Weiterhin wird aufgezeigt, welcher Jitter und welche Latenzzeiten mit diesen Technologien erreicht werden können.**

### **Verfügbare Technologien**

Grundsätzlich gibt es zwei Ansätze, um Linux echtzeitfähig zu machen:

- Den sogenannten Mikrokern-Ansatz und
- den sogenannten In-Kernel-Ansatz

Im Mikrokern Ansatz werden alle Echtzeitaufgaben in einem eigenen RTOS gehandhabt, Linux wird innerhalb dieses RTOS als niederpriorer Task gescheduled. Genaugenommen muss hier also nicht von Echtzeit mit Linux, sondern vielmehr von Echtzeit neben Linux gesprochen werden.

Der sogenannte In-Kernel Ansatz verfolgt das Ziel, Linux an sich echtzeitfähig zu machen (ohne darunterliegenden Mikrokern). Im Folgenden sollen verschiedene Vertreter dieser Ansätze vorgestellt werden.

### **RTAI**

Das Realtime Application Interface (RTAI) ist eine Entwicklung der Technischen Universität Mailand und entstand unter der Schirmherrschaft von Professor Paolo Mantegazza. RTAI ist ein klassischer Vertreter des Mikrokern Ansatzes. Oberstes Designziel von RTAI ist und war es, die kleinstmöglichen Latenzzeiten auf einer gegebenen Hardwareplattform zu erzielen. Dieses Designziel bedingt diverse Einschränkungen für RTAI Applikationen. Weiterhin wird nur eine recht kleine Anzahl an Zielplattformen unterstützt (derzeit x86, x86\_64 und diverse ARM Plattformen). In der Praxis finden sich kaum noch neue Projekte, die auf RTAI aufsetzen. Abbildung 1 zeigt den prinzipiellen Aufbau von RTAI.

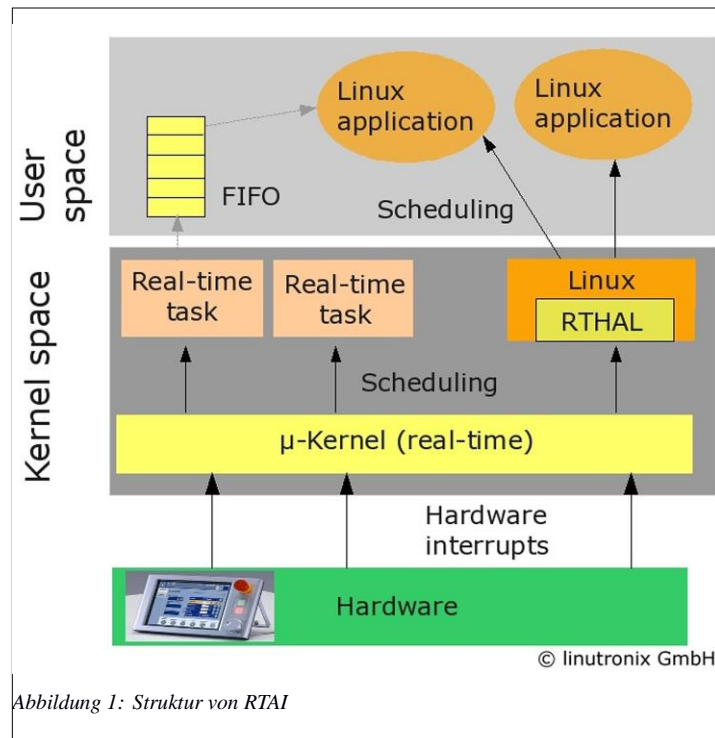


Abbildung 1: Struktur von RTAI

## Xenomai

Das Xenomai Projekt wurde im Jahre 2001 gegründet. Im Gegensatz zu RTAI erlaubt es Xenomai Echtzeit im Userspace relativ einfach zu nutzen (RTAI erlaubt dies nur sehr eingeschränkt). Die Besonderheit von Xenomai sind die sogenannten Skins, die es vereinfachen sollen, Applikationen von anderen Echtzeitsystemen (z. Bsp. VxWORKS, ...) nach Xenomai zu portieren. Xenomai Skins bilden die API dieser Systeme ab. Xenomai unterstützt derzeit folgende Architekturen: PowerPC32, PowerPC64, x86, x86\_64, Blackfin, ARM und ia64). Die zentralen Begriffe im Designkonzept von Xenomai stellen Xenomai Nucleus, die Interrupt Pipeline (IPIPE), Hardware Abstraction Layer (HAL) und System Abstraction Layer (SAL) dar. Die IPIPE kann bildlich als virtueller Interruptcontroller betrachtet werden. Sie organisiert das System in verschiedene Domains. Interrupts werden von IPIPE entgegengenommen und an die einzelnen Domains verteilt. Nucleus beinhaltet die Xenomai Core Funktionalität. Diese ist zuständig dafür, alle notwendigen Ressourcen bereitzustellen, die Skins benötigen um die Funktionalität von RTOS-en nachzubilden zu können. Der Hardware Abstraction Layer beinhaltet den Plattform und CPU abhängigen Code. Alle darüber liegenden Layer (darunter auch Nucleus) bauen darauf auf. Sowohl im Kernel als auch in der Applikation muss mit eigenen Bibliotheken und einer eigenen API gearbeitet werden. Es können also nicht die Standard Linux Bibliotheken verwendet werden! (dies gilt auch für RTAI). Abbildung 2 zeigt das Konzept von Xenomai.

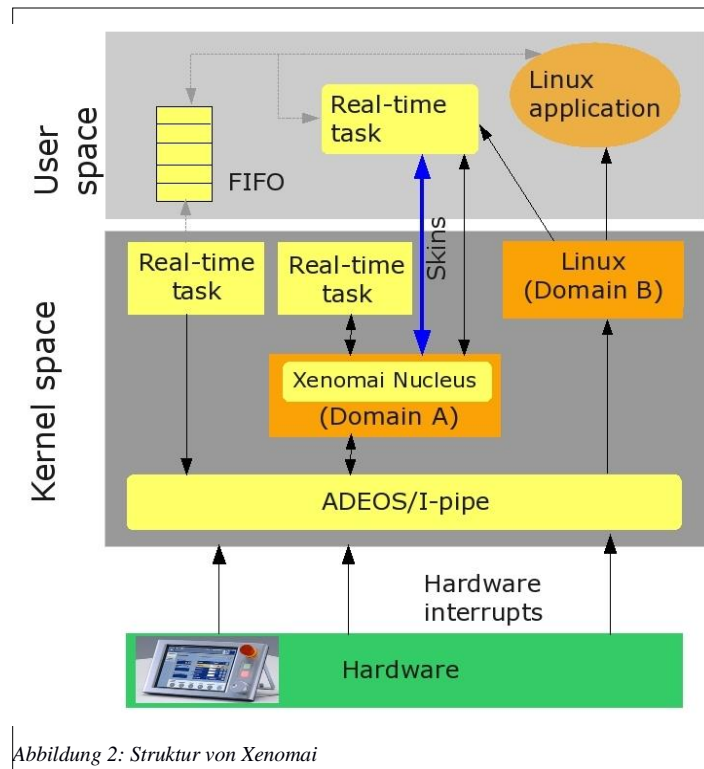


Abbildung 2: Struktur von Xenomai

### Der Realtime Preemption Patch (PREEMPT\_RT)

Der Realtime Preemption Patch entstand ursprünglich aus Arbeiten von Ingo Molnar und Thomas Gleixner. Vor allem Thomas Gleixner ist heute die treibende Kraft bei der Entwicklung von PREEMPT\_RT. Im Gegensatz zu RTAI und Xenomai macht PREEMPT\_RT den Linux Kernel an sich echtzeitfähig. Dies wird im Besonderen durch folgende Mechanismen erreicht:

- Sleeping Spinlocks: Spinlocks werden durch RT Mutexe ersetzt. Raw Spinlocks ersetzen die Eigenschaft der ursprünglichen Spinlocks
- Threaded Interrupt Handlers: Interrupt Handler laufen per Default nicht im harten Interruptkontext, sondern als Kernelthread.

Viele Mechanismen, die ursprünglich in PREEMPT\_RT entwickelt wurden, haben schon lange Ihren Weg in den Mainline Linuxzweig gefunden: High Resolution Timer (Hochauflösende Timer unabhängig vom Scheduler Tick), Priority Inheritance, generisches Interrupthandling für alle Architekturen und bereits in 2.6.30 die Threaded Interrupt Handler. Weiterhin hat sich die Linux Entwicklergemeinschaft schon im Jahre 2006 darauf geeinigt, dass Preempt RT in den Linux Kernel integriert wird. Mit den Arbeiten des OSADL und einer neuen Working Group der Linux Foundation (RTL, gegründet Oktober 2015; Gründungsmitglieder sind u.a. Google, OSADL, Intel, ARM, Texas Instruments, Altera, National Instruments u.a.) wurde erst kürzlich ein weiterer großer Schritt in diese Richtung getan. Aufgrund der vielen Vorteile und der großen Akzeptanz in der Linux Community hat sich PREEMPT\_RT in den letzten Jahren als de-facto Standard für Echtzeitlinux durchgesetzt. Neben den vielen technologie-bedingten Vorteilen ist noch die Tatsache erwähnenswert, dass sich das OSADL in umfangreichem Maße mit der Qualitätssicherung für PREEMPT\_RT basierte Systeme befasst. Hierfür werden in einer Testfarm unzählige Benchmarks

und Latenzzeitmessungen auf einer Vielzahl von Hardware durchgeführt (<https://www.osadl.org/QA-Farm-Realtime.qa-farm-about.0.html>).

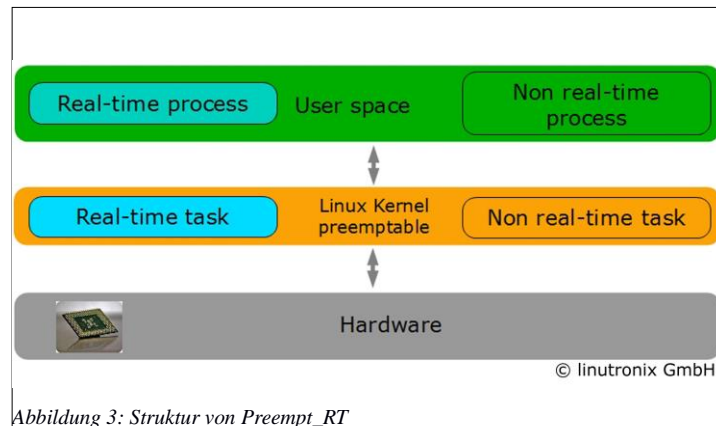


Abbildung 3: Struktur von Preempt\_RT

Weiterhin bietet der Realtime Preemption Patch den großen Vorteil, dass Echtzeitapplikationen als POSIX Realtime Applikationen geschrieben werden können. Es wird keine spezielle API verwendet. PREEMPT\_RT unterstützt eine Vielzahl von Architekturen (PowerPc, x86, x86\_64, MIPS, ARM, ...).

Wie Abbildung 3 zeigt, integriert PREEMPT\_RT die Echtzeitfunktionalität "nahtlos" in den Linux Kernel. Auch die Entwickler anderer Projekte haben die Vorzüge von PREEMPT\_RT bereits erkannt. Xenomai 3 bietet Unterstützung für PREEMPT\_RT. Dies ermöglicht den Einsatz von Xenomai Skins auf PREEMPT\_RT Kernen.

### Evaluierung der unterschiedlichen Ansätze

Zur Gegenüberstellung von Mikro-Kernel und In-Kernel Ansätzen wurden vergleichende Messungen auf einer **ARM Cortex A9 CPU** durchgeführt. Xenomai wurde als Vertreter der Mikro-Kernel Technologie gewählt, PREEMPT\_RT als Vertreter der In-Kernel Technologie. Gemessen wurde die Reaktionszeit auf ein externes Signal, welches mit einer Frequenz von 10kHz generiert wurde. Zur Durchführung der Messungen wurde die OSADL Latency Box verwendet (<https://www.osadl.org/uploads/media/OSADL-Latency-Box.pdf>). Gemessen wurde jeweils die Reaktionszeit für Kernel und Applikation. Alle Messungen wurden unter denselben Bedingungen und 100% CPU Last durchgeführt (erzeugt mit dem Programm „hackbench“).

## Ergebnisse Xenomai

Abbildung 4 zeigt die Reaktionszeit auf das Event im Kernel:

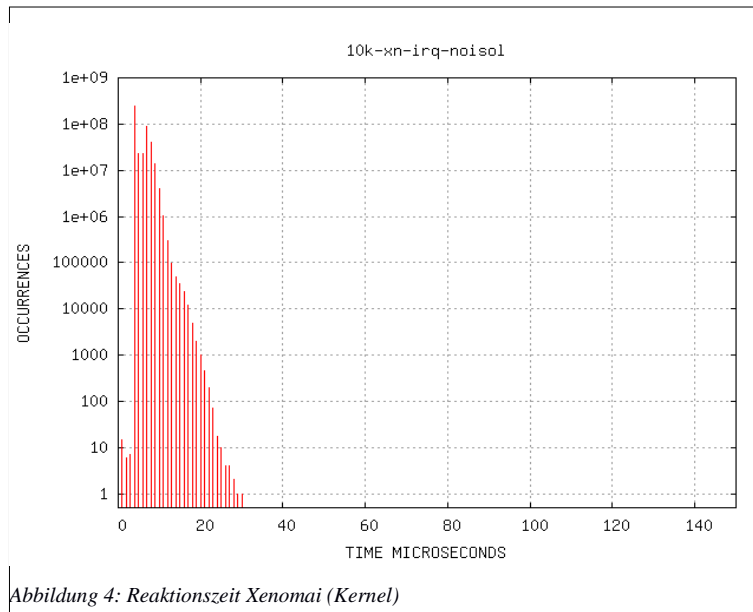


Abbildung 4: Reaktionszeit Xenomai (Kernel)

Abbildung 5 stellt die Latenzzeiten für eine Applikation dar, die auf das Event wartet. Der Worst-Case liegt bei knapp unter 100 Mikrosekunden.

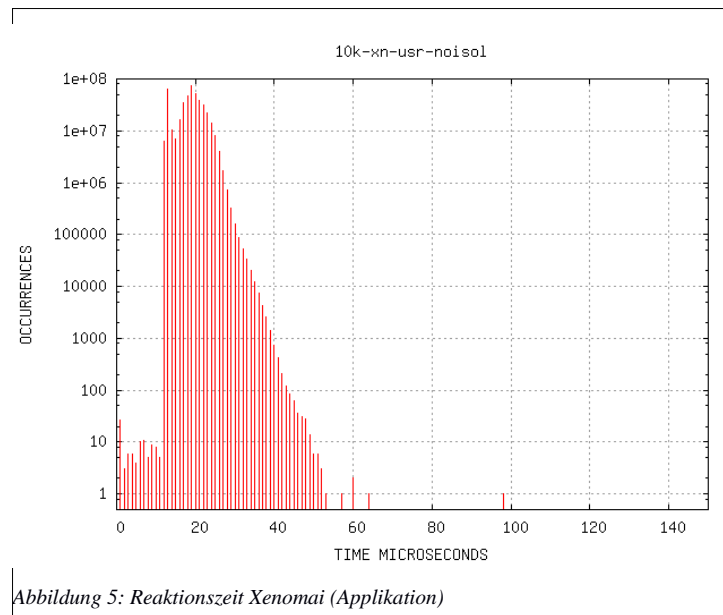
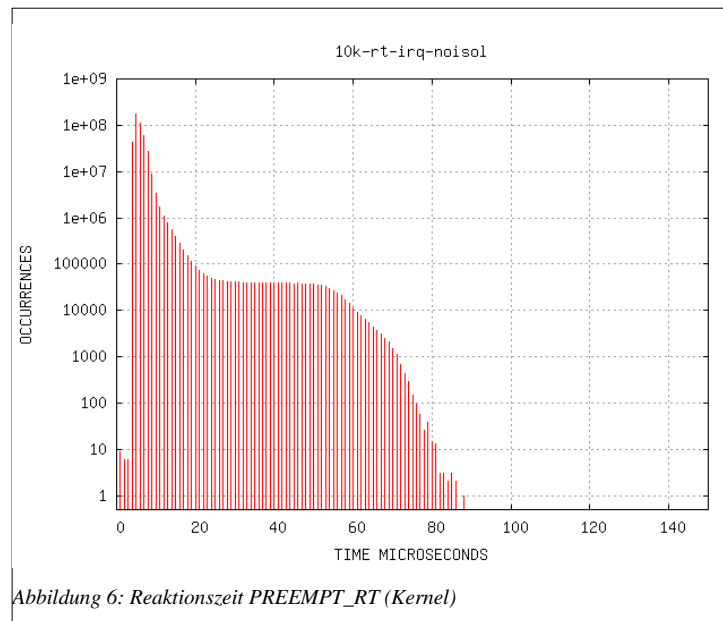


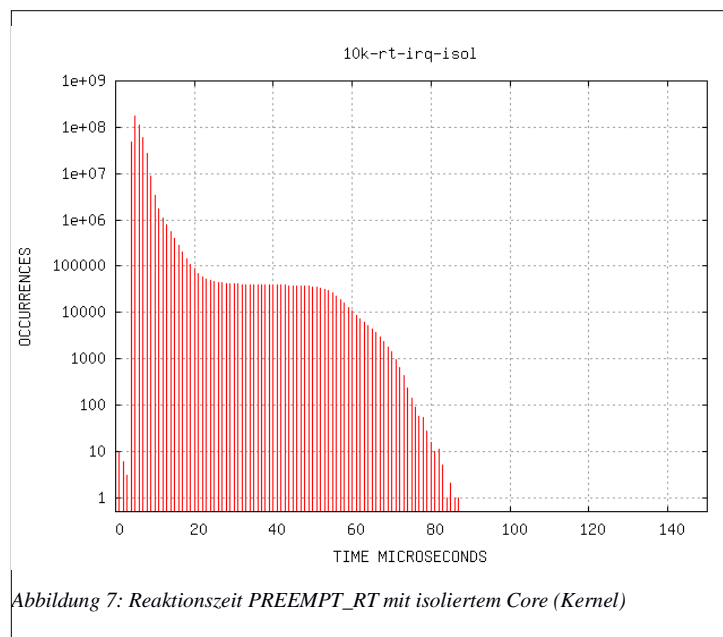
Abbildung 5: Reaktionszeit Xenomai (Applikation)

## Ergebnisse PREEMPT\_RT

Abbildung 6 + 7 zeigen die Reaktionszeiten im Kernel unter PREEMPT\_RT.



Durch Isolieren eines Cores lässt sich der Worst-Case noch etwas verbessern.



Bei den Messungen für die Applikation schneidet PREEMPT\_RT etwas besser ab als Xenomai. Der Worst-Case liegt bei etwas über 90 Mikrosekunden.

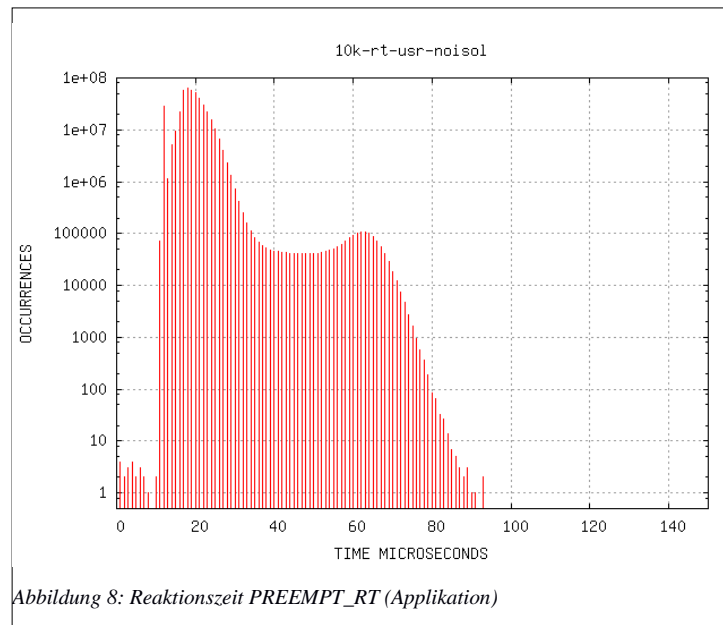


Abbildung 8: Reaktionszeit PREEMPT\_RT (Applikation)

Durch Isolieren eines Cores lässt sich das Ergebnis auf 80 Mikrosekunden verbessern:

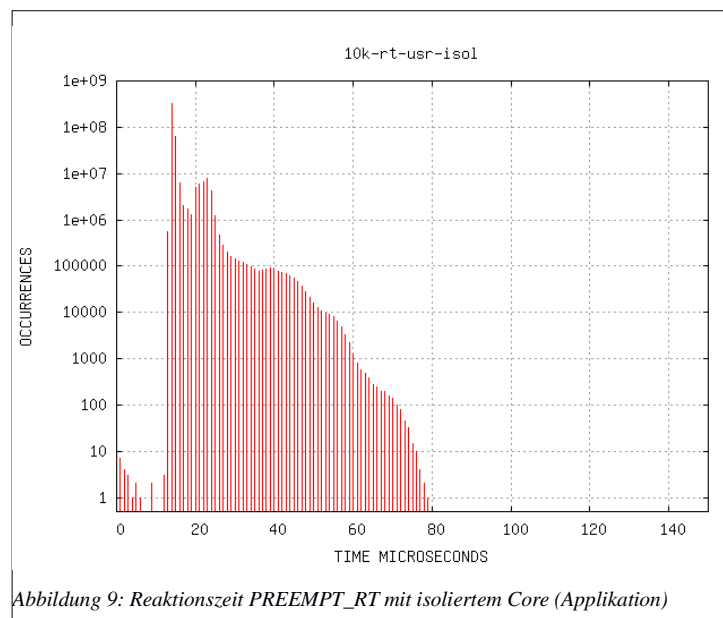


Abbildung 9: Reaktionszeit PREEMPT\_RT mit isoliertem Core (Applikation)

Nun ist der Vergleich der Reaktionszeiten im Kernel nicht ganz fair, denn bei Xenomai sprechen wir hier ja von einem Mikrokern und nicht vom Linux Kernel. Denn genau genommen arbeiten wir den Code ja nicht im Linux Kontext ab (wie es bei Xenomai der Fall ist). Daher wurde für PREEMPT\_RT noch eine weitere Messung durchgeführt: Das Abarbeiten des kritischen Codepfades im FIQ Kontext (den viele

ARM basierte SOCs bieten). Der FIQ lebt in seiner „eigenen Welt“, es ist aber möglich einen FIQ Handler aus Linux heraus zu registrieren!

Abbildung 10 zeigt die Ergebnisse einer FIQ basierten Lösung (basierend auf einem PREEMPT\_RT Kernel):

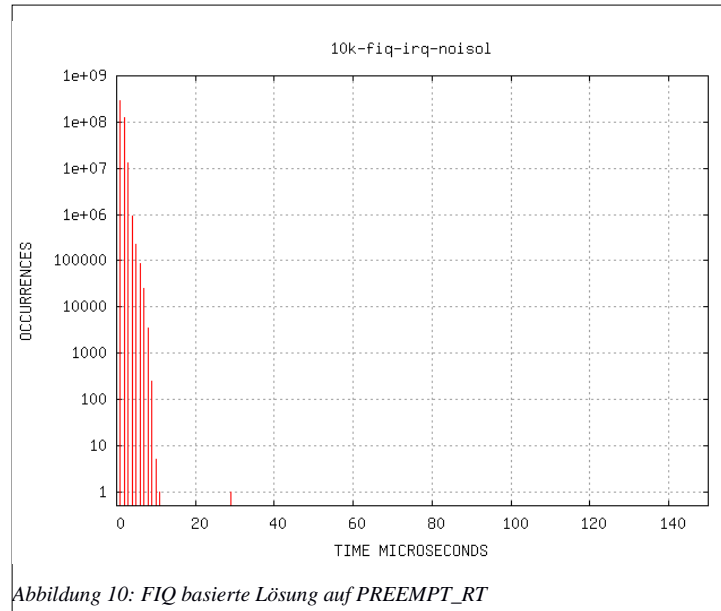


Abbildung 10: FIQ basierte Lösung auf PREEMPT\_RT

Der Worst-Case ließ sich hier auf 30 Mikrosekunden verbessern! Dieser einzelne „Ausreißer“ lässt sich mit großer Wahrscheinlichkeit auf ein Hardwareproblem zurückführen. Auf anderen Plattformen konnten mit diesem Ansatz Reaktionszeiten von < 10us erreicht werden!

### Fazit

Linux besitzt mit der passenden Erweiterung hervorragende Echtzeiteigenschaften. Aufgrund der hohen Akzeptanz in der Entwicklergemeinde und der einfachen Handhabbarkeit hat sich hierfür der sogenannte PREEMPT\_RT Ansatz als Standard etabliert. Die Latenzzeiten dieses In-Kernel Ansatzes sind auf Applikationsebene vergleichbar mit denen von Mikrokernen (wie Xenomai). Die Mikrokern können lediglich im Kernel bessere Latenzen erreichen, wobei hier zu berücksichtigen ist, dass hier nicht im Linux Kontext gearbeitet wird, sondern im Mikrokern (und somit auch mit dessen Restriktionen und API). Wer bereit ist, für bessere Latenzzeiten Restriktionen in Kauf zu nehmen, kann auf ARM basierten Systemen auch mit einer FIQ Lösung und PREEMPT\_RT arbeiten. Hiermit sind teilweise Latenzzeiten unter 10 Mikrosekunden zu möglich.

PREEMPT\_RT bietet in Summe den besten Tradeoff aus geringen Latenzzeiten und einfacher Handhabbarkeit. Weiterhin ist zu beachten, dass Organisationen wie das OSADL die Qualität des PREEMPT\_RT Ansatzes kontinuierlich prüfen. Eine weitere gute Nachricht ist die Tatsache, dass die Linux Foundation das RTL Projekt ins Leben gerufen hat, das die Integration in den Linux Mainlinekernel über die nächsten Jahre finanziert und vorantreibt. Diese Tatsachen geben dem Anwender zusätzliche Planungssicherheit für zukünftige Projekte und die Pflege bestehender Produkte!