HAW HAMBURG

Masterarbeit

Johannes Reidl

Evaluation potenzieller Echtzeitbetriebssysteme einer CNC-Steuerung

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Johannes Reidl

Evaluation potenzieller Echtzeitbetriebssysteme einer CNC-Steuerung

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 25. Januar 2022

Johannes Reidl

Thema der Arbeit

Evaluation potenzieller Echtzeitbetriebssysteme einer CNC-Steuerung

Stichworte

RTOS, Xenomai, Preempt_RT, CNC, FPGA, PCI-E, DMA, Interrupt, Linux Device Driver

Kurzzusammenfassung

Commercial of-the-shelf Hardware in Verbindung mit einem linuxbasierten Betriebssystem findet immer häufiger Anwendung in industriellen Echtzeitsystemen. Dies bringt viele Herausforderungen für die Echtzeit mit sich, wie etwa schwer vorhersagbares Cachingverhalten oder Service Management Interrupts. In dieser Arbeit werden mögliche alternative Betriebssysteme unter Berücksichtigung besagter Herausforderungen für die Echtzeit untersucht. Hierzu werden aus dem komplexen Aufbau einer CNC-Steuerung Anforderungen an das Betriebssystem sowie der Portierungsaufwand abgeleitet und mögliche Kandidaten auf ihr Echtzeitverhalten untersucht.

Johannes Reidl

Title of Thesis

Evaluation of potential realtime operating systems for a CNC-controller

Keywords

RTOS, Xenomai, Preempt_RT, CNC, FPGA, PCI-E, DMA, Interrupt, Linux Device Driver

Abstract

Commercial of-the-shelf Hardware in conjunction with Linux-based operating systems are increasingly used in industrial realtime systems. New challenges for the realtime arise from this, like hard to predict caching-behaviour or Service Management Interrupts. In this work there will be a survey of potential alternative operating systems with regards

to the before mentioned challenges for the realtime. The requirements for the potential new operating systems and the necessary porting efforts will be derived from the complex structure of a CNC-controller software. The realtime behavior of the candidates for the new operating systems will also be determined.

Inhaltsverzeichnis

1 Einleitung

Motivation

Elektronische Geräte sind aus einem modernen Haushalt nicht mehr wegzudenken. Spülmaschine, Mixer, Staubsauger und Co. sind längst zum Standard geworden. Auch der Trend hin zu Smart Homes ist ungebrochen. Rund 44 Millionen Haushalte in Europa waren im Jahr 2020 mit Smarthomegeräten ausgestattet und für 2025 wird eine Anzahl von 97,2 Millionen prognostiziert[1]. Ebenso nimmt an vielen anderen Stellen unseres täglichen Lebens die Anzahl elektronischer Geräte zu, z.B. Steuergeräte für die Fahrassistenz und den Komfortbereich in Pkws[31]. Besonders signifikant ist diese Zunahme jedoch bei Smartphones. Im Jahr 2020 betrug die Anzahl der Personen mit einem Smartphone weltweit 6,06 Milliarden[2]. Ubiquitious Computing, also die Omnipräsenz von Computern in unserem Leben, ob offen sichtbar oder versteckt, z.B. eingearbeitet in Kleidung, ist längst zur Realität geworden [25][19].

Für all diese elektronischen Geräte müssen die Leiterplatten gebohrt und gefräst werden. Dies geschieht mit sogenannten CNC Maschinen (Computerized Numerical Control). Diese Maschinen sind in der Lage, automatisch und mit hoher Präzision eine große Anzahl an Werkstücken zu fertigen. In der Industrie sind der Grad der Präzision, also die Güte des fertigen Werkstücks, sowie eine hohe zeitliche Effizienz die maßgeblichen kompetitiven Kriterien [5]. Um dies zu erreichen, ist es nötig, die Servomotoren der CNC Maschine mit exaktem Timing anzusteuern. Bereitgestellt wird dieses exakte Timing durch die Steuerung der CNC Maschine. Die Firma Sieb & Meyer[3], mit Firmensitz in Lüneburg, ist Weltmarktführer im Bereich der CNC Steuerungen. Um die zunehmenden Anforderungen

[1] https://de.statista.com/infografik/23403/geschaetzte-anzahl-der-smart-homes/

[2] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide

[3] https://www.sieb-meyer.de

und Kundenwünsche zu erfüllen, wurde von der Sieb & Meyer AG die Produktlinie CNC-9X vor 14 Jahren ins Leben gerufen. Diese Steuerung ist komplexer und besitzt mehr Features sowie eine stärkere Hardware als die Vorgängerreihe der CNC-8X. Handelt es sich bei der Hardware der CNC-8X noch um eine maßgeschneiderte eingebettete Lösung, gefertigt und entwickelt im eigenen Haus, so wurde diese bei der CNC-9X gegen kommerziell erhältliche Module ersetzt (COTS – Commercial off-the-shelf). Dieser Wechsel zu gewöhnlicher Hardware mit erhöhter Rechenleistung ist bei Cyber-Physical-Systems in den letzten Jahren häufig zu beobachten. Selbst bei Systemen mit hohen Anforderungen an die funktionale Sicherheit und Zuverlässigkeit, bei denen für gewöhnlich eingebettete Systeme zum Einsatz kommen [16]. Die Steuerung der CNC-8X läuft ohne Betriebssystem. Bei ihrem Nachfolger wird ein Echtzeitbetriebssystem auf Basis des Linuxkernels verwendet. Auf der einen Seite werden dadurch neue, komplexe Probleme, wie das Einhalten garantierter Latenzen, eingeführt. Auf der anderen Seite verbessert sich dadurch die „Time to Market“, die Rechenleistung sowie die spätere Updatefähigkeit beträchtlich [24]. Auch Debugging wird durch die Verfügbarkeit vieler gut dokumentierter und weit verbreiteter Tools wie etwa des GNU Debuggers GDB[4] stark erleichtert.

Das Echtzeitbetriebssystem der CNC-9X wurde zuletzt vor sieben Jahren aktualisiert. Dadurch ist die Plattform sehr gut erschlossen, die Kompatibilität mit der Steuerungssoftware garantiert und das Maß an nötiger Wartung verschiedener Revisionen gering. Dennoch ergeben sich mittlerweile einige Nachteile aus der Verwendung dieses in die Jahre gekommenen Betriebssystems:

- Die Unterstützung für die Komponenten neuer CPU-Module fehlt.
- Es können keine aktuellen Versionen der Entwicklungstoolchain verwendet werden.
- Neu hinzugekommene Funktionen des Betriebssystems können nicht verwendet werden.
- Fehler im verwendeten Linux Kernel und des Betriebssystems bleiben ungeschlossen.

Aus den oben genannten Gründen soll daher nun das Echtzeitbetriebssystem der CNC Steuerung durch ein aktuelleres ersetzt werden. Die Untersuchung alternativer Echtzeitbetriebssysteme unter Berücksichtigung ihres Echtzeitverhaltens ist Gegenstand dieser Arbeit. Die genauen Ziele und die Rahmenbedingungen sind in den beiden nachfolgenden Abschnitten aufgeführt.

[4] https://www.gnu.org/software/gdb

Zielsetzung

Das Ziel dieser Arbeit wird wie folgt definiert:

Es sollen mögliche Nachfolger für das derzeitig verwendete Echtzeitbetriebssystem Xenomai 2 der CNC Steuerung CNC95.00 gesucht und der Portierungsaufwand eingeschätzt werden. Hierzu muss die Architektur der CNC Steuerung ermittelt und ihr Echtzeitverhalten untersucht werden. Die Interruptlatenz potentieller Nachfolger soll unter anderem mit dem Oszilloskop gemessen werden. Darüber hinaus soll eine Aufwandsabschätzung für eine Portierung auf die neue Plattform erfolgen. Abschließend sollen die Ergebnisse evaluiert werden.

Rahmenbedingungen

Diese Arbeit wird für die Sieb & Meyer AG, ansässig *Auf dem Schmaarkamp 21, 21339 Lüneburg*, in der Abteilung „Entwicklung CNC Software“ durchgeführt. Es gilt, die nachfolgend aufgeführten Rahmenbedingungen zu beachten.

- Es gibt verschiedene Basisboards der Steuerung und daher verschiedene Hardware Revisionen. Untersucht werden soll das zum Zeitpunkt dieser Arbeit aktuelle Board.
- Soweit möglich, soll die Steuerungssoftware wegen Wahrung von Betriebsgeheimnissen als Blackbox betrachtet werden.
- Messungen, die Rückschlüsse auf Interna der Software zulassen würden, werden abstrahiert bzw. vereinfacht dargestellt.
- Es können keine Untersuchungen des Quellcodes durchgeführt werden.
- Mögliche alternative Betriebssysteme müssen frei verfügbar sein.
- Die Steuerungssoftware wird kontinuierlich weiterentwickelt, erhält neue Features und Bugfixes. Für die Analyse der Software wird ein eingefrorener Snapshot verwendet.

Struktur der Arbeit

Strukturieren lässt sich diese Arbeit wie folgt: Das folgende Kapitel 2 führt die nötigen Grundlagen ein. Zuerst werden die verschiedenen Arten der Echtzeitsysteme aufgeführt und Begriffe wie die „Worst Case Execution Time“ eingeführt. Als nächste Punkte werden das Scheduling und die Speicherverwaltung des Linux Kernels erklärt. Ebenso wird eine kurze Einführung in Linux Gerätetreiber gegeben. Danach wird das Scheduling unter Echtzeitbetriebssystemen untersucht. Darauf folgt ein Überblick über die Herausforderungen bei Echtzeitsystemen und Commercial-of-the-shelf Hardware. Hierbei werden zunächst einige Aspekte der Intel Atoms der Bay Trail Serie erläutert. Unter anderem wird die Interruptverarbeitung untersucht sowie die CPU Speicher Caches und Performance Register. Abschließend wird auf das Messen der Performanz von Echtzeitbetriebssystemen eingegangen. In diesem Zusammenhang werden die Programme „Perf“ und „Cyclictest“ erläutert.

Nachdem die Grundlagen dargelegt wurden, wird in Kapitel 3 der Ist-Zustand untersucht. Zunächst wird der allgemeine Aufbau der Hardware vorgestellt und der Zusammenhang der Komponenten erläutert. Als nächstes wird die Steuerungssoftware einer Analyse unterzogen. Hierbei wird das Caching- bzw Branching-Verhalten der Software untersucht, ebenso wie das Echtzeitverhalten. Zuletzt widmet sich das Kapitel dem aktuell verwendeten Betriebssystem Xenomai 2 und untersucht Kenngrößen wie die Latenzen der Interruptverarbeitung sowie des Schedulings.

Die Suche nach einem Nachfolger für den Xenomai 2 wird in Kapitel 4 durchgeführt. Zuerst werden die Anforderungen an das neue Betriebssystem aus dem Ist-Zustand abgeleitet. Ausgehend davon werden mögliche Alternativen für das Betriebssystem ermittelt. Die CNC Steuerungssoftware wurde an 32Bit angepasst. Ein möglicher Umstieg auf 64Bit und die daraus resultierenden Konsequenzen werden diskutiert. Untersucht wird ebenfalls der Portierungsaufwand für die Kandidaten.

In Kapitel 5 werden die im vorhergehenden Kapitel ermittelten Kandidaten einer genaueren Prüfung ihrer Echtzeitqualitäten unterzogen. Dazu wird zunächst das „State of the Art“ Vorgehen untersucht. Danach wird die Scheduling Latenz mit Cyclictest ermittelt. Abschließend folgt die Messung der Interruptlatenz mit dem Oszilloskop. Eine Auswertung und Diskussion aller Ergebnisse erfolgt in Kapitel 6. Zuletzt werden die Ergebnisse noch einmal in Kapitel 7 zusammengefasst und ein Ausblick auf die Zukunft gegeben.

2 Grundlagen

Diese Arbeit setzt einige Grundlagen voraus, welche in den nachfolgenden Abschnitten dargelegt werden. Zuerst wird der Begriff Echtzeit eingeführt und erläutert, was ein Echtzeitsystem ausmacht. Auch der Begriff der Worst Case Execution Time wird eingeführt. Danach wird in Abschnitt 2.2 ein Überblick über Linux gegeben. Hierzu werden das Scheduling und die Speicherverwaltung erläutert. Abschließend folgt eine kurze Einführung in Linux Gerätetreiber. In Abschnitt 2.3 werden Echtzeitbetriebssysteme im Kontrast zu konventionellen Betriebssystemen vorgestellt und das Scheduling erläutert.

2.1 Echtzeitsysteme

Echtzeitsysteme werden bereits seit Jahrzehnten untersucht. So haben bereits 1990 Stankovic und Ramamritham Echtzeitsysteme definiert und Kriterien zur Kategorisierung solcher Systeme in [30] zusammengetragen. Demnach ist ein Echtzeitsystem ein System, bei dem seine Korrektheit nicht nur von den logischen Ergebnissen ihrer Berechnungen abhängt, sondern auch von der Zeit zu der diese Ergebnisse produziert werden. Stankovic und Ramamritham identifizieren fünf Hauptkriterien zur Kategorisierung:

1. Wie schnell muss das System reagieren können?
2. Wie streng sind die Deadlines?
3. Wie zuverlässig muss das System sein?
4. Was ist der Umfang des Systems und was ist der Grad der Interaktion unter den Komponenten?
5. Wie verhält sich die Umgebung in der das System läuft?

Aus diesen Kategorien abgeleitet haben sich unter anderem die folgenden Echtzeitbegriffe entwickelt. Serino et al. [27] sprechen von „weicher“ und „harter“ Echtzeit. Reghenzani et al. [24] ergänzen den Begriff der „festen“ Echtzeit. Brown et al. [4] verwenden die Einteilung in „weiche“, „lebenssicherheit harte“, „100% harte“ und „95% harte“ Echtzeit. Als letzte Ergänzung sei noch die Einführung des Begriffs der„wahrscheinlichen harten“ Echtzeit durch Bernet et al. [2] erwähnt.

Weiche Echtzeit Bei weicher Echtzeit geht es darum, einen QoS aufrecht zu erhalten, wie etwa bei einem Audio/Video Stream. Werden ein paar Deadlines nicht eingehalten, führt dies zu einer Verschlechterung der Qualität, jedoch ohne eine Gefahr für Personen oder Gegenstände.

Feste Echtzeit ist ähnlich der weichen Echtzeit. Eine verpasste Deadline macht jedoch die Berechnung bzw. das Ergebnis unbrauchbar und es muss verworfen werden.

Harte Echtzeit bezeichnet die strengte Klasse von Echtzeit. Es darf unter keinen Umständen eine Deadline verpasst werden, da dies zu unerwünschten Konsequenzen führen kann.

100% harte Echtzeit entspricht zusammen mit der Lebenssicherheit harten Echtzeit der Harten Echtzeit. 100% harte Echtzeit bezieht nur möglichen Schaden für Dinge ein, jedoch nicht Personen.

Lebenssicherheit harte Echtzeit entspricht zusammen mit der 100% harten Echtzeit der Harten Echtzeit. Sie beinhaltet auch Systeme, bei denen das Verpassen einer Deadline zu einer Gefahr für den Menschen führen kann.

95% harte Echtzeit ist ein Begriff für Systeme, die harte Echtzeitanforderungen besitzen, es jedoch verkraftbar ist, wenn gelegentlich ($\leq 5\%$ der Fälle) Daten unbrauchbar sind.

Wahrscheinliche harte Echtzeit verfeinert das Konzept der 95% harten Echtzeit. Anstatt eine feste Grenze von 5% zu definieren, wird eine für das System verkraftbare Anzahl an verpasster Deadlines definiert, die mit einer bestimmten Wahrscheinlichkeit eintreffen.

Ein wichtiger Begriff bei Echtzeitsystemen ist die „Worst Case Execution Time“ (WCET). Um sicherzustellen, dass eine Deadline auch wirklich eingehalten werden kann, muss die maximale Laufzeit eines Echtzeittasks bekannt sein. Diese maximale Laufzeit wird als WCET bezeichnet. Laut Reghenzani et al.[24] ist die WCET die typische Metrik für

einen Echtzeittask. Es gibt jedoch viele Einflussfaktoren auf die WCET: Das Scheduling des Betriebssystems, Caching und Branching, Interruptverarbeitung etc. Die wichtigsten dieser Einflussfaktoren werden im weiteren Verlauf, insbesondere in Abschnitt 2.4, erläutert.

2.2 Linux

Seit seiner Veröffentlichung 1991 hat sich GNU+Linux zu einem ausgereiften Betriebssystem entwickelt, welches vom kleinsten Elektronikspielzeug bis hin zum Supercomputer in unzähligen Geräten zum Einsatz kommt [36]. Auch für diese Arbeit spielt Linux eine wichtige Rolle. Insbesondere das Verständnis des Scheduling, der Speicherverwaltung und Gerätetreiber sind für den weiteren Verlauf wichtig. Diese sollen im nachfolgenden Abschnitt näher erläutert werden.

2.2.1 Scheduler

Seit der Kernel Version 2.6.23 ist der „Completely Fair Scheduler“ (CFS) der Standard Prozess Scheduler des Linux Kernels[1]. Der Scheduler setzt das Konzept der „virtuellen Laufzeit“ um. Für jeden Task in den Zuständen *runnable* und *blocked* wird die virtuelle Laufzeit notiert. Je niedriger die virtuelle Laufzeit eines Tasks, desto höher rutscht dieser in der Prioritätenliste und wird dementsprechend vom CFS höher in der Queue eingefügt.

Der Completely Fair Scheduler implementiert folgende drei Policies:

SCHED_NORMAL Diese Scheduling Policy wird für gewöhnliche Tasks verwendet.

SCHED_BATCH Wird nicht annähernd so oft unterbrochen, als das bei gewöhnlichen Tasks der Fall wäre. Erlaubt es Tasks länger zu laufen und Caches besser auszunutzen, jedoch auf Kosten der Interaktivität. Gut geeignet für batch Aufgaben.

SCHED_IDLE Nur wenn die CPU sonst im Idle Modus wäre, wird der Task ausgeführt.

[1] https://www.kernel.org/doc/html/latest/scheduler/index.html

Ebenfalls seit der Kernel Version 2.6 besitzt Linux Echtzeitfähigkeiten[2] und Scheduler, um Echtzeittasks zu verwalten. Diese Echtzeitscheduler, „SCHED_FIFO“, „SCHED_-RR“ und „SCHED_DEADLINE“ werden in Abschnitt 2.3 genauer erläutert. Hinzugekommen sind die Kernel Konfigurationen „CONFIG_PREEMPT_VOLUNTARY“ und „CONFIG_PREEMPT“. CONFIG_PREEMPT_VOLUNTARY fügt im Linux Kernel an Stellen, die bekannt dafür sind lange zu blockieren, Überprüfungen ein, ob ein höher priorisierter Task ausgeführt werden möchte und unterbricht dann den Programmfluss zu Gunsten dieses Tasks. Dies eliminiert jedoch nicht die Möglichkeit, dass solche Stellen mit langer Latenz auftreten. Bei CONFIG_PREEMPT sind alle Bereiche des Kernels, abgesehen von Spin-Lock geschützten Bereichen und Interrupt-Handlern, unterbrechbar. Dies reduziert laut RT-Wiki die Scheduling Latenz in den Bereich einstelliger Millisekunden. Dennoch besteht die Gefahr, dass Gerätetreiber sehr hohe Interruptlatenzen besitzen und somit unvorhersehbar hohe Latenzen auftreten können.

2.2.2 Speicherverwaltung

Dieser Abschnitt beschäftigt sich damit, wie Linux Speicherchips unter einer x86 Mikroprozessor Architektur verwendet [3]. Über Speicheradressen kann der Inhalt einer Speicherzelle ausgelesen werden. Hier muss jedoch im Fall einer x86 Architektur unter vier Arten von Adressen unterschieden werden.

Logische Adresse Ist in den Befehlen der Maschinen enthalten, um die Adresse eines Operanden oder einer Instruktion anzugeben. Jede logische Adresse besteht aus einem Segment und einem Offset, welches den Abstand vom Beginn des Segments zur eigentlichen Adresse angibt. Logische Adressen bilden den Standard Adressraum im Kernel. Sie bilden eine bestimmte Menge an Hauptspeicher in den Kernelspace ab und werden oft so behandelt, als wären es physikalische Adressen. Alle logischen Adressen sind virtuelle Kernel Adressen, aber viele virtuelle Kernel Adressen sind keine logischen Adressen. Dies ist darauf zurückzuführen, dass virtuelle Adressen nicht zwingend eine 1:1-Beziehung zu physikalischen Adressen besitzen, wie dies bei logischen Adressen der Fall ist. Dieser Zusammenhang ist auch auf der rechten Seite der Abbildung 2.1 zu sehen.

[2] https://rt.wiki.kernel.org

Virtuelle Adresse Wird auch Lineare Adresse genannt und ist eine einzelne 32Bit bzw. 64Bit unsigned Integer, je nach CPU Architektur. Adressen in einem Userspace Programm sind virtuelle Adressen.

Physikalische Adresse Wird dazu verwendet, um Speicherzellen in Speicherchips anzusprechen. Sie entsprechen den elektrischen Signalen, die entlang der Adresspins des Mikroprozessors an den Speicher Bus gesendet werden.

Bus Adresse Sind die Adressen zwischen Peripherie Bussen und dem Hauptspeicher. Moderne PCI Busse können z.B. als Busmaster fungieren und somit auch über den Adress/Datenbus Werte in den Hauptspeicher schreiben. Dies wird als Direct Memory Access (DMA) bezeichnet.

Wie sich die Adressierung in Relation zum physikalischen Speicher verhält, ist in Abbildung 2.1 zu sehen. In der Mitte ist der physikalische Speicher abgebildet. Er ist aufgeteilt in „high“ und „low“ Memory. Low Memory bezeichnet Speicher, für den logische Adressen im Kernelspace existieren. Für gewöhnlich ist der gesamte Speicher Low Memory. High Memory ist dementsprechend Speicher, für welchen keine logischen Adressen existieren, da er jenseits des Adressraums liegt, welcher für virtuelle Kernel Adressen reserviert ist.

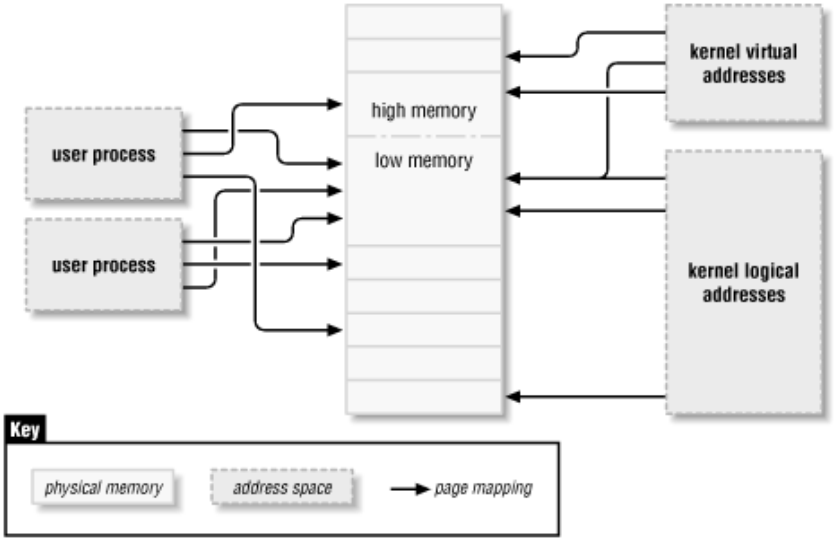


Abbildung 2.1: Adresstypen in Linux[14]

Durch eine sogenannte Memory Management Unit (MMU) können logische Adressen in virtuelle Adressen unter Verwendung einer Hardwarekomponente namens Segmentation Unit umgewandelt werden. Durch eine weitere Hardwarekomponente, Paging Unit

genannt, können virtuelle Adressen in physikalische Adressen umgewandelt werden. An dieser Stelle genügt es zu wissen, dass es verschiedene Adressräume gibt, je nachdem ob man sich im Kernel- oder Userspace befindet sowie für Peripherie und dass sich die Adressen in andere Adressräume übersetzen lassen. Detailwissen über diese Hardwarekomponenten ist für das Verständnis dieser Arbeit nicht nötig.

2.2.3 Gerätetreiber

Eine der Komponenten der CNC Steuerung ist ein Linux Gerätetreiber für eine PCIe Karte. Hier sollen nun die Grundzüge für einen solchen Treiber erläutert werden. Als Grundlage dient das Standardwerk „Linux Device Drivers, Third Edition von Jonathon Corbet, Alessandro Rubini und Greg Kroah-Hartman[14]“.

Es gibt verschiedene Möglichkeiten, um Gerätetreiber zu klassifizieren. Die gebräuchlichste ist die im Folgenden beschriebene Klassifizierung. Aus der Sicht des Linux Kernels betrachtet existieren drei grundlegende Arten von Gerätetreibern: Block-, Charakter- und Netzwerk-Treiber. In der Regel implementiert ein Treiber mindestens eines dieser Interfaces. Der Treiber der CNC Steuerung ist ein Charakter-Treiber, daher beschränkt sich die Betrachtung auf diese Art der Module.

Auf ein Charakter (char) Gerät kann über einen Bytestrom, ähnlich wie bei einer Datei, zugegriffen werden. Dazu werden vom char-Treiber Dateisystemknoten, vergleichbar einer Datei, wie etwa /dev/tty und /dev/lp0 angelegt. Um darauf zugreifen zu können, muss der Treiber in der Regel mindestens die System Calls *open*, *close*, *read* und *write* implementieren. Der einzige relevante Unterschied eines char-Treiberknotens und einer Datei ist, dass beim Lesen und Schreiben in einer Datei immer vor und zurück gesprungen werden kann, wohingegen die meisten Char-Geräte nur Datenkanäle sind, auf welche lediglich sequentiell zugegriffen werden kann. Dennoch existieren auch Char-Treiber, bei denen vor und zurück gesprungen werden kann. Bei Videograbbern ist dies z.B. der Fall.

Ein Gerätetreiber läuft im Adressraum des Kernels, im sogenannten Kernelspace. Dies unterscheidet ihn von einer gewöhnlichen Anwendung, welche im Userspace läuft. Um mit einem Treiber zu interagieren, gibt es das System Call Interface. Über dieses Interface, bereitgestellt durch die „ioctl“ Methode, können Daten ausgetauscht werden oder Aktionen im Treiber ausgelöst werden. Dazu muss die Userspace Anwendung über einen Dateizugriff auf den Treiberknoten entsprechende Kommandos senden. Wird ein System

Call ausgeführt oder tritt ein Hardware Interrupt auf, transferiert Linux die Ausführung vom Userspace in den Kernelspace.

PCI Treiber

PCI Geräte werden beim Systemstart konfiguriert. Ein Treiber für PCI/PCIExpress Hardware muss in der Lage sein, diese Konfigurationsinformationen auszulesen, um das Gerät zu initialisieren. Laut der Dokumentation des Linux Kernels[3] müssen für die Initialisierung die nachfolgenden Schritte ausgeführt werden:

- finde und aktiviere das Gerät
- fordere Zugriff auf MMIO/IOP Ressource an
- setze die DMA-Masken Größe
- alloziere und initialisiere geteilte Kontrolldaten
- greife auf den Konfigurationsadressraum des Gerätes zu
- registriere einen IRQ Handler
- initialisiere nicht-PCI Teile des Chips
- aktiviere DMA

Die DMA-Masken Größe beträgt nach der PCIExpress Spezifikation[22] für alle PCI-X und PCIe konformen Geräte 64Bit. Ursprünglich konnte nur die CPU als Busmaster fungieren. Moderne PCI Bus Architektur ist jedoch auch dazu in der Lage. Dazu enthalten moderne PCs Zusatzhardware in Form eines DMA Controllers, welcher Daten zwischen dem RAM und dem I/O-Gerät austauscht. Somit findet der Datentransfer ohne die CPU statt. Zuvor muss der Treiber des Gerätes jedoch sicherstellen, dass der DMA Controller direkten Zugriff auf die Busadressen hat, in denen die Daten liegen. Diese Busadressen müssen zuerst vom Kernel in den physikalischen Adressraum der CPU konvertiert werden. Damit der Treiber auf diese Ressourcen zugreifen kann, müssen die physikalischen Adressen noch einmal in den virtuellen Adressraum des Kernels gemappt werden.

Das PCIExpress Gerät der CNC Steuerung unterstützt MSI-X Interrupts. Diese werden in Unterabschnitt 2.4.1 genauer beschrieben. Dem PCIe Gerät wurde beim Bootvorgang

[3] https://www.kernel.org/doc/html/latest/PCI/pci.html

eine IRQ Nummer zugeteilt. Für diesen Interrupt muss ein sogenannter Handler, die Interrupt Service Routine (ISR), implementiert werden. Unter Linux muss diese zwingend im Treiberkontext umgesetzt werden. Ein Interface für den Userspace ist nicht vorhanden. Tritt ein Interrupt auf, so wird die ISR ausgeführt. Dabei ist darauf zu achten, dass die ISR „so kurz wie möglich“ gestaltet wird, um die Zeit, in der Interrupts deaktiviert sind, gering zu halten.

2.3 Scheduling in Echtzeitbetriebssystemen

Echtzeitbetriebssysteme unterscheiden sich von gewöhnlichen Betriebssystemen dahingehend, dass sie in der Lage sein müssen, auf ein Ereignis in angemessener Zeit zu reagieren.[27] Diese Zeit hängt davon ab, von welcher Art die Echtzeitanforderung, wie sie in Abschnitt 2.1 beschrieben wurde, ist. Ein Echtzeitbetriebssystem muss also gewährleisten können, dass ein Task mit hoher Priorität zum richtigen Zeitpunkt ausgeführt wird. Daher unterscheidet sich die Art und Weise wie Tasks gescheduled werden. Nachfolgend werden übliche Scheduler, die Echtzeitanforderungen berücksichtigen, vorgestellt.

Ein Echtzeitbetriebssystem verwendet im Scheduler ein Prioritätensystem und Möglichkeiten, niedrig priorisierte Tasks zu unterbrechen. Häufig wird als Standardmethode das sogenannte first-in-first-out Scheduling, dargestellt in Abbildung 2.2 verwendet.

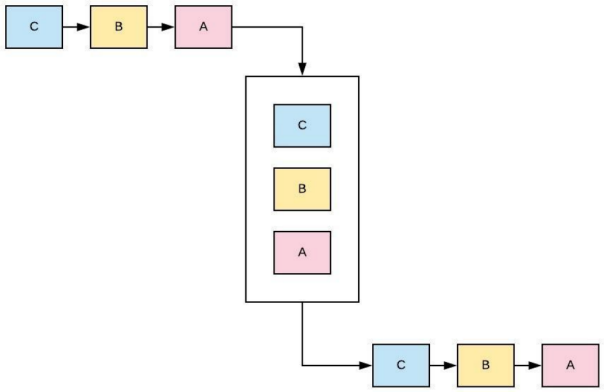


Abbildung 2.2: FIFO Scheduling[27]

Bei dieser Methode versucht der Scheduler den am höchsten priorisierten Task auszuführen. Dieser läuft, bis er fertig abgearbeitet wurde oder ein noch höher priorisierter Task die CPU benötigt. Besitzen mehrere Tasks die selbe Priorität, so werden diese in

der Reihenfolge wie sie in die Scheduling Queue eingetragen wurden, abgearbeitet. Im Bild werden zuerst der Task A, dann der Task B und schließlich der Task C in die Queue eingetragen und in der selben Reihenfolge ausgeführt.

Die in Abbildung 2.3 skizzierte Methode Round-robin behandelt gleich priorisierte Tasks auf andere Weise. Besitzen mehrere Prozesse die selbe Echtzeit Priorität, so wird der Task, welcher an erster Stelle in der lokalen Queue des CPU-Kerns steht, ausgeführt, bis seine Zeitscheibe aufgebraucht wurde.[3] Danach wird er unterbrochen und der andere Task läuft ebenso lange, bis seine Zeitscheibe aufgebraucht wurde.

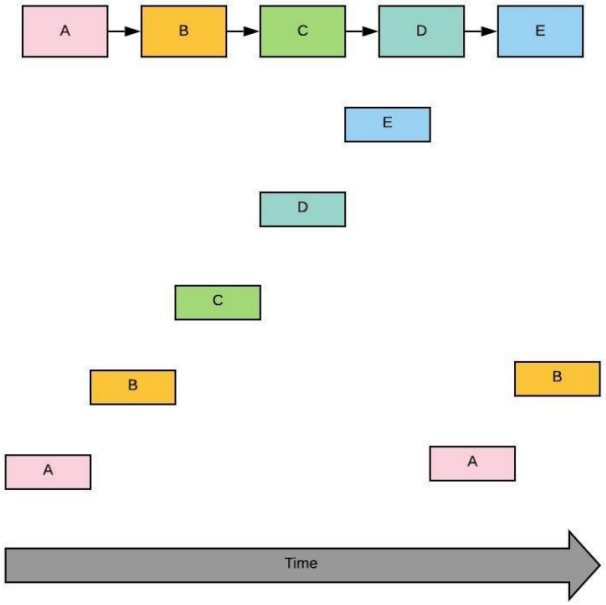


Abbildung 2.3: Round-robin Scheduling[27]

Im Bild zu sehen sind die fünf Tasks A bis E mit der selben Priorität. Zuerst wird Task A ausgeführt, bis er seine Zeitscheibe aufgebraucht hat und wird dann zu Gunsten von Task B unterbrochen. Dies wird solange fortgesetzt bis alle Tasks ihre erste Zeitscheibe verbraucht haben und der Zyklus mit A von neuem beginnt oder keiner der Tasks mehr Rechenzeit benötigt.

Die scheduling Technik „Earliest Deadline First“ berechnet die Priorität eines Tasks dynamisch. Sie ist in Abbildung 2.4 zu sehen. Sie berücksichtigt dabei die Ankunftszeit des Tasks, die benötigte Laufzeit und die Deadline. Am höchsten priorisiert wird dann die Task mit der frühesten Deadline.

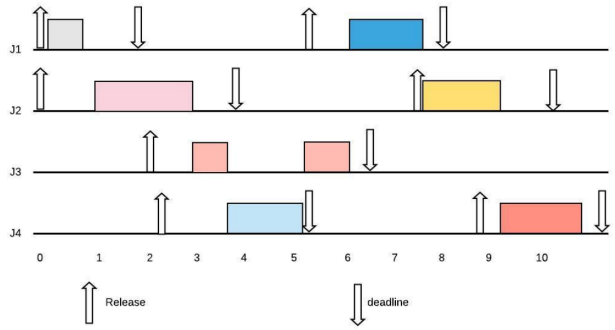


Abbildung 2.4: Earliest Deadline First Scheduling[27]

Die Pfeile nach oben im Bild mit der Beschriftung release bedeuten, dass ab hier der Task zur Ausführung bereit ist. Im Bild zu sehen ist unter anderem, dass der Scheduler alle Tasks so gelegt hat, dass sie sich nicht überschneiden und manche in der Ausführung verzögert wurden, um den anderen mit dringlicheren Deadlines Rechenzeit zu verschaffen. Ebenso zu sehen ist, dass der Task J3 von dem blauen Task J4 unterbrochen wird, da dessen Deadline dringender ist und für die Ausführung von J3 noch mehr Zeit übrig ist.

2.4 Herausforderungen bei Commercial off-the-shelf Systemen und Echtzeit

Per Design sind Commercial off-the-shelf Systeme nicht für (harte) Echtzeitbetriebssysteme ausgelegt. Anstelle von Vorhersagbarkeit sind die Ausführungsgeschwindigkeit und der Durchsatz priorisiert. Dennoch finden sie aus Kostengründen, erhöhter Flexibilität und schneller Time-to-Market zunehmend Einzug in industriellen Anwendungen [24].

In diesem Abschnitt werden nun Eigenschaften solcher Plattformen untersucht, die Einfluss auf die Latenzen und somit die Echtzeiteigenschaften haben können.

2.4.1 Interruptverarbeitung auf Intel Plattformen

Durch Interrupts können signifikante Latenzen in einem System entstehen [33]. Die Plattform, auf der die CNC Steuerungssoftware läuft, basiert auf einer Intel Architektur und beinhaltet einen Atom Prozessor der Bay Trail Serie. Für diese Plattform soll nachfolgend die Interruptverarbeitung im Echtzeitkontext untersucht werden.

Interrupt Ein Interrupt ist ein Hardwaremechanismus, der dazu verwendet wird, die CPU zu informieren, dass ein asynchrones Ereignis eingetreten ist [15]. Wurde ein solches Ereignis registriert, wird vom Betriebssystem die Interrupt Service Routine (ISR) angesprungen, nachdem die CPU den aktuellen Kontext gespeichert hat. Innerhalb dieser Routine wird der Interrupt verarbeitet und bei Vollendung quittiert. Danach wird der Kontext vor dem Interrupt wiederhergestellt. Jeder CPU Kern der Intel Atom E3800 Familie besitzt einen Interruptcontroller. Somit ist die CPU in der Lage, vier solcher Interrupts parallel zu verarbeiten[12]. Es ist möglich, Interrupts zu deaktivieren und wieder zu aktivieren. In einem Echtzeit Kontext sollten Interrupts so wenig wie möglich deaktiviert werden, da dies zu höherer Interrupt Latenz führen kann, ebenso wie zu verpassten Interrupts[15]. Auch können Interrupts verschachtelt werden, sollte während der Verarbeitung eines Interrupts ein weiterer, wichtigerer Interrupt auftreten.

Nicht-Maskierbarer Interrupt Die Mikroprozessor Familie der E3800 besitzt einen sogenannten Nicht-maskierbaren Interrupt (NMI). Diese werden verwendet, wenn eine Aufgabe schneller abgearbeitet werden muss, als das bei der Verarbeitung durch einen Kernel der Fall wäre. Sie besitzen eine höhere Priorität als gewöhnliche Interrupts. Ein NMI kann nicht deaktiviert werden, daher ist seine Latenz minimal. NMIs spielen für gewöhnlich im normalen Betrieb keine Rolle und werden in der Regel bei Ereignissen wie System Resets, System Debugging oder nicht wiederherstellbaren Hardwarefehlern verwendet.

System Management Interrupt System Management Interrupts (SMI) stellen eine dritte, potentielle Störungsquelle für Echtzeitanwendung dar. Sie versetzen die CPU in den System Management Mode (SMM) und besitzen die höchste Priorität unter den Interrupts. In diesem Modus ist es der Firmware des Motherboards möglich, die CPU zu beliebigen Zeiten zu unterbrechen und Low-Level Programme auszuführen. Dies ist z.B. häufig bei Lüftersteuerungen der Fall, aber auch Zugriffe auf andere Hardware ist möglich. SMIs sind schwer oder gar nicht vorhersagbar und können vom Betriebssystem nicht bemerkt werden. Ebenso fügen sie dem Betriebssystem signifikante Latenzen hinzu. Macarenco et al. haben den Einfluss von SMIs auf die Latenz in ihrer Arbeit[20] untersucht und kommen zu dem Ergebnis, dass durch längere SMIs merkbare Verzögerungen in der Ausführungszeit von Programmen mit mehreren Threads auftreten. Auf der von Sieb & Meyer verwendeten Plattform wurden durch den Hersteller daher SMIs in der Firmware des Motherboards deaktiviert.

MSI und MSI-X Message Signaled Interrupts stellen die dritte Generation von Interrupt Methoden für I/O-Geräte dar[6]. Sie erhöhen die Anzahl an der zur Verfügung stehenden Interrupts auf 224 im Gegenzug zu den 24 der Vorgängergeneration und bis zu 2048 bei MSI-X. Seit 2004 sind MSI Interrupts fester Bestandteil der PCIe Spezifikation[22]. Interrupt Sharing ist hier nicht mehr erlaubt. Stattdessen wird bei MSI der Interrupt Vektor durch das PCIe Gerät direkt in den Lokalen-APIC (Advanced Programable Interrupt Controller) geschrieben. Eine Verarbeitung des Interrupts findet ohne Überprüfung und Quittierung statt. Dadurch reduziert sich die Interruptlatenz bei MSI deutlich.

2.4.2 CPU Speicher Caches

Schneller Speicher ist teuer, daher folgen in modernen CPUs Speicher einem hierarchischen Modell, welches in Abbildung 2.5 dargestellt ist. Die Grafik zeigt typische Speichergrößen für Desktop PCs und Laptops. Hierbei gilt: Je näher sich der Speicher an der CPU befindet, umso kleiner und schneller ist er und umso mehr kostet er pro Speichereinheit. Die Register in der CPU bilden die oberste Schicht und besitzen in der Regel eine Größe von 1000 bytes für Laptops und 2000 bytes für Desktop PCs. Ihre Zugriffsgeschwindigkeit liegt bei 300ps. Wird mehr Speicher benötigt, gibt es zwei bis drei Schichten Caches. Der Level 1 Cache ist häufig 64KB groß mit einer Geschwindigkeit von 1ns, der Level 2 Cache 256KB mit 3-10ns und der Level 3 Cache 4-8MB mit 10-20ns. Danach kommt der Hauptspeicher, der nicht mehr Teil der CPU ist und daher über einen Bus mit dieser verbunden ist. Übliche Größen für den Hauptspeicher sind 4-16GB für Laptops und 8-64GB für Desktops. Seine Zugriffszeiten liegen bereits bei 50-100ns. Den größten Speicher bildet die Festplatte oder ein Flashspeicher mit 256GB bis 2TB mit einer deutlich höheren Zugriffszeit von 50-100µs für den Flashspeicher und ca. 5-10ms für die Festplatte.

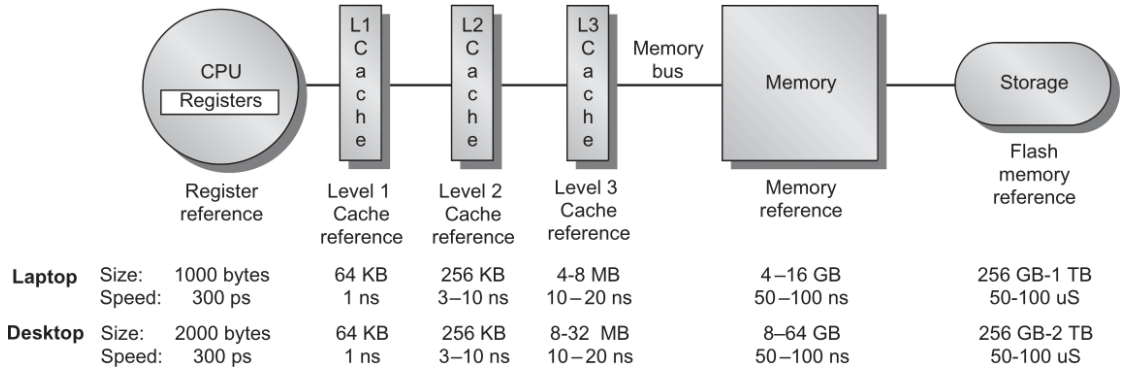


Abbildung 2.5: Speicherhierarchie für Laptop und Desktop[11]

Den größten Einflussfaktor auf die Performanz einer CPU bilden die Caches[11]. Versucht ein Programm auf Daten aus dem Cache zuzugreifen und diese sind vorhanden, spricht man hier von einem Cache Hit. So werden, je nach Level des Caches, nur wenige CPU Zyklen benötigt. Sind sie nicht vorhanden, so handelt es sich um einen Cache Miss. Dieser kann im Falle eines Misses im Last Level Cache in der Größenordnung mehrerer 100 Zyklen liegen[18]. Ist ein Cache Miss aufgetreten, so wird solange in der nächsten, darunter liegenden Schicht nachgefragt, bis die gewünschten Daten vorhanden sind. Es gilt, je höher die Anzahl an Hits, desto größer die Performanz, wie Abbildung 2.6 zeigt. Die Differenz in der Performanz ist zwischen 98% und 99% viel größer als zwischen 10% und 11%. Dieses nichtlineare Profil ergibt sich aus den unterschiedlichen Geschwindigkeiten bei Hits und Misses. Je größer der Unterschied, desto steiler wird die Steigung.

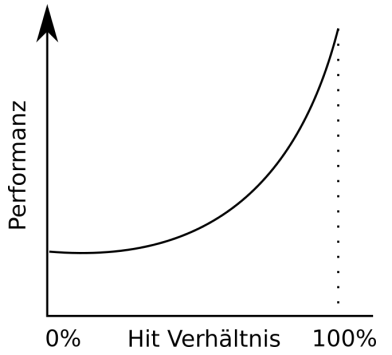
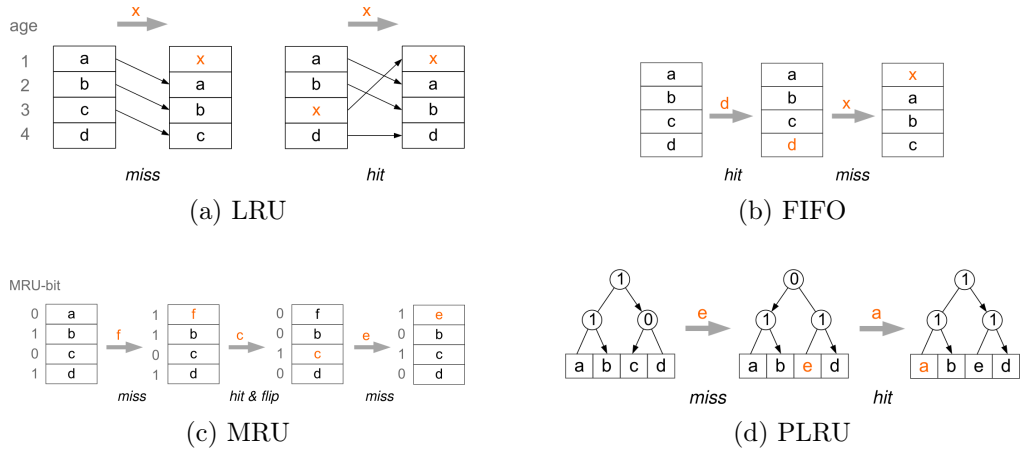


Abbildung 2.6: Cache Hit Verhältnis und Performanz[10]

Cache Adressierung wird häufig „Set-Assoziativ“ organisiert. Hierbei wird der Cache in voneinander unabhängige Sets aufgeteilt. Jeder Cache hat eine bestimmte Anzahl an Wegen, von denen jeder einer einzelnen Cache Line eines Cache Sets zugeordnet ist. Die Gesamtzahl der Wege eines Cache Sets werden Assoziativität genannt. Um einen Speicherblock zu laden, muss die CPU zuerst feststellen, zu welchem Set dieser Block gehört. Danach wird im Zielset nach einem freien Cache Weg gesucht. Sind alle Wege belegt, so tritt eine Ersetzungsstrategie in Kraft, die festlegt, welcher

alte Block durch den neuen ersetzt wird. Ein paar dieser Ersetzungsstrategien werden nachfolgend betrachtet.



(a) LRU (b) FIFO

(c) MRU (d) PLRU

Abbildung 2.7: Häufige Cache Ersetzungsstrategien[18]

LRU: Steht für Least Recently Used, also was am längsten nicht verwendet wurde. Jedem Cache Weg ist ein festes Alter zugewiesen, welches das Alter des entsprechenden Blocks festlegt. In Abbildung 2.7a ist das Verhalten des Caches bei einem Hit bzw. Miss abgebildet, wenn X angefordert wird. Im Falle eines Misses wird X an erster Stelle eingetragen, da es am aktuellsten verwendet wurde. Der Rest wird nach unten verschoben. Da d am längsten nicht verwendet wurde, wird es aus dem Cache entfernt. Bei einem Hit wird der Cache neu geordnet, da nun X als letztes verwendet wurde und somit an oberster Stelle steht.

FIFO: First in first out. Im Falle eines Hits wird nicht neu geordnet wie bei LRU und im Falle eines Misses, wird, wie in Abbildung 2.7b zu sehen, X an oberster Stelle eingetragen und der Rest nach unten verschoben.

MRU: Most Recently Used. Hier ist jedem Cache Weg ein Bit zugeordnet, welches eine Abschätzung liefert, wie lange der letzte Zugriff her ist. Eine 1 bedeutet, dass der Cache Weg vor kurzem benutzt wurde. Nach einem Hit wird das Bit auf 1 gesetzt. Bei einem Miss wird der oberste Weg, dessen Bit 0 ist, durch das neue Datum ersetzt und das Bit auf 1 gesetzt. Wird irgendwann der letzte Weg mit einer 0 angefordert, so wird sein Bit auf 1 gesetzt und alle anderen Bits der anderen Wege auf 0 gesetzt. Dies wird globaler Flip genannt. Zu sehen ist dies in Abbildung 2.7c, wenn c angefordert wird.

PLRU: LRU in Hardware zu implementieren ist sehr komplex und benötigt verhältnismäßig viel Energie. PLRU (Pseudo LRU) ist eine Baum-basierte Annäherung an LRU. Hier werden die Cache Wege an den Blättern eines Binärbaums mit k-1 bits angeordnet, wobei k die Assoziativität ist. Bit 0 und Bit 1 bezeichnen den rechten und den linken Unterbaum. Folgt man den Bits von der Wurzel bis nach unten, erhält man die zu ersetzende Cache Line. In Abbildung 2.7d wird zuerst e angefordert. Der oberste Knoten ist 1, also wird dem rechten Pfad gefolgt. Der zweite Knoten ist 0, somit wird die linke Line, das c, ersetzt. Nach einem Hit und nach einem Miss werden alle Knoten auf dem genommenen Pfad so gesetzt, dass sie von der gerade genommenen Line weg zeigen.

Auch können durch die sogenannte Cache-Kohärenz Latenzen entstehen. Es ist möglich, dass Speicher in verschiedenen Caches auf verschiedenen CPU-Kernen zur gleichen Zeit gecached wird. Wird dieser Speicher nun durch einen Prozessor verändert, so müssen alle Caches upgedatet werden. Dies wird Cache-Kohärenz genannt. Dadurch können im LLC Zugriffsverzögerungen auftreten, wie nachfolgend aufgeführt[17].

- LLC Hit, Line nicht geteilt: ∼40 CPU Zyklen
- LLC Hit, Line geteilt in einem anderen Kern: ∼65 CPU Zyklen
- LLC Hit, Line geändert in einem anderen Kern: ∼75 CPU Zyklen

Durch eine Cache Analyse kann die Genauigkeit der WCET abgeschätzt werden. Hierbei muss eine statische Analyse des Cachingverhaltens stattfinden und versucht werden, mit dem Ergebnis die Anzahl der Hits vorherzusagen. Je genauer die Vorhersage der Hits, desto genauer ist die Einschätzung der WCET[18].

Häufig ist das verwendete Cachingmodell jedoch nicht dokumentiert oder aus anderen Gründen nicht verfügbar. Abel et al. etwa haben 2014 in ihrer Arbeit [1] für einige zu dieser Zeit aktuellen Prozessoren die Cachingstrategien durch aufwändiges Reverse Engineering ermittelt. Eine statische Analyse des Cachingverhaltens der CNC Steuerung ist aus dem oben genannten Grund nicht möglich und würde den Rahmen dieser Arbeit sprengen. Ebenso steht der Quellcode der CNC Steuerung für eine Betrachtung dieser Art nicht zur Verfügung. Dennoch kann eine dynamische Analyse stattfinden und damit eine Abschätzung der Laufzeiten der CNC Steuerung erfolgen.

2.4.3 Sonstige Herausforderungen bei COTS Hardware

Es gibt noch weitere Einflussfaktoren bei COTS Hardware auf die Echtzeit. Da es nur wenige bis überhaupt keine Möglichkeiten gibt, auf diese Einfluss zu nehmen werden sie hier nur am Rande erwähnt.

Moderne x86 CPUs besitzen neben den Caches noch weitere Eigenschaften, die es schwierig machen, einen deterministischen Programmfluss zu erzeugen und die Ausführungsgeschwindigkeit zum Teil kontextabhängig machen[7].

Pipelining Instruktionen durchlaufen in einem modernen Prozessor eine mehrstufige Pipeline. Ein einfaches Beispiel für eine Pipeline besitzt vier Stufen. In der ersten Stufe wird die Instruktion geladen, in der zweiten die Instruktion dekodiert und Daten geladen, in der dritten wird der Befehl ausgeführt und in der vierten Stufe schließlich das Ergebnis zurückgegeben. Eine Pipelintiefe von 20 und mehr Stufen ist für aktuelle Mikroprozessoren nicht ungewöhnlich[11].

Dynamic Branch Prediction Um eine möglichst gute Performanz eines Prozessors mit Pipelinig zu erreichen, muss die Pipeline immer gefüllt sein. Im Falle von if-then-else Anweisungen, also bei Sprüngen im Programmfluss, muss geraten werden, welche Abzweigung genommen wird. Bei dynamischer Branch Prediction, also Vorhersage, werden zur Laufzeit Informationen über zuvor genommene Abzweigungen gesammelt und versucht, damit eine Vorhersage zu treffen. Wird eine Abzweigung falsch vorhergesagt, so werden die Instruktion, welche weiter unten in diesem Pfad bereits in der Pipeline sind, verworfen.

Out-of-Order Execution In einfachem Pipelinig wird In-Order Execution verwendet. Das bedeutet, dass die Instruktionen in der Reihenfolge des Programmcodes abgearbeitet werden. Wird die Pipeline blockiert, etwa durch eine falsche Branch Vorhersage, dann kann keine weitere Instruktion ausgeführt werden, bis das Problem behoben ist. Bei der Out-of-Order Execution kann die Ausführung der Instruktionen von der durch das Programm vorgegebenen Reihenfolge abweichen, ohne jedoch das Ergebnis zu beeinflussen. Hier können mehrere Befehle parallel ausgeführt werden, sobald ihre Daten verfügbar werden.

Superskalarität Bezeichnet die Eigenschaft eines Prozessors mit mehreren physikalischen Kernen, Instruktionen aus einem Strom parallel zu verarbeiten.

2.5 State of the Art

In Abschnitt 2.3 wurde bereits eine allgemeine Einführung in Echtzeitbetriebssysteme gegeben. Hier soll untersucht werden, welche Metriken für die Ermittlung der Echtzeitgüte in der Literatur herangezogen werden und welche Studien bereits durchgeführt wurden.

2.5.1 Akademische Literatur

Als Einstieg dient die Arbeit [24] von Reghenzani et al. Hier wird der State of the Art von Arbeiten, welche Echtzeitfähigkeiten in den Linux Kernel integrieren, untersucht. Der Schwerpunkt liegt auf dem Preempt_RT Patch.

Laut [24] kann die Response Time eines Tasks auf einer nicht vollständig deterministischen COTS Plattform mit Linux ausgedrückt werden als eine zufällige Variable X, welche einer unbekannten Verteilung $X \sim \mathcal{A}$ folgt. Unter einer gegebenen Deadline D wird ein Task als ein harter Echtzeittask angesehen, wenn er folgende Gleichung erfüllen muss:

$$P(X \leq D) = 1 \tag{2.1}$$

Das ist in der Realität schwer zu erreichen. Ebenso schwer, unter einer Plattform wie der beschriebenen, ist es, dies formal zu beweisen. Daher wird der weniger strikte Fall $P(X \leq D) < 1$, mit $E[X] = \mu$ und $VAR[X] = \sigma^2$. Wobei E den Erwartungswert und VAR die Varianz bezeichnet. Wird nun angenommen, dass Verbesserungen der Latenzen im Kernel zu einer Verteilung X' führt, so dass $\mu' < \mu$, können Reghenzani et al. daraus keine Schlüsse für die Echtzeitperformanz ziehen.

$$\mu' < \mu \nRightarrow P(X' \leq D) \geq P(X \leq D). \tag{2.2}$$

Rückzuführen ist dies darauf, dass die Reduktion des durchschnittlichen Wertes nicht zwangsweise zu einer besseren Abschätzung führt: Es kann sein, dass $\sigma' > \sigma$ und daher

kann der obere Wert der Abschätzung der modifizierten Variante größer sein als der des Originals. Selbst bei einer Analyse, in der $\sigma' \leq \sigma$ gilt, kann der rechte Teil der Gleichung 2.2 nicht angenommen werden:

$$\mu' < \mu, \sigma' < \sigma \not\Rightarrow P(X' \leq D) \geq P(X \leq D). \tag{2.3}$$

Dies gilt, da σ die Standardabweichung repräsentiert und nicht den Jitter, der stattdessen die maximale Abweichung angibt. Ein Abschätzungsintervall kann nicht dazu verwendet werden, um Ausreißer und deren Maxima zu erkennen, die dazu benötigt werden, um die WCET zu ermitteln. Reghenzani et al. schlussfolgern aus der Schwierigkeit, Ergebnisse aus Gleichung 2.1 zu bekommen, der Schwäche aus Gleichung 2.2 und Gleichung 2.3 sowie der Unausgereiftheit der Wahrscheinlichkeitstheorie für Echtzeit, dass dies die Hauptursachen für das Fehlen formaler Demonstration der Echtzeitgüte im Linux Kernel sind. Ebenso kommen sie zu dem Ergebnis, dass viele Untersuchungen dazu angestellt werden, die Latenz zu verringern, ohne zu beweisen, dass dies nicht zu einer Erschwerung der Unberechenbarkeit führt. Dies führt nach der Meinung der Autoren dazu, dass Schlüsse aus der maximal beobachteten WCET gezogen werden müssen, welche lediglich eine untere Schranke darstellt. Die Bestimmung der oberen Schranke auf Anwendungsebene verlangt einen unrealistischen Aufwand. Patterson und Hennessy ergänzen hierzu in [11]: „Nimmt man an, dass alle Branches falsch vorhergesagt werden und alle Cachezugriffe ins Leere gehen, ist die WCET übermäßig pessimistisch.“

Reghenzani et al. untersuchen des Weiteren die Methodik und Metriken, welche für die Bestimmung der Echtzeitgüte verwendet werden. Zunächst kann festgestellt werden, dass es bereits einige Untersuchungen in der Literatur gibt, jedoch ein Vergleich schwer fällt, da sich die verwendeten Hardwareplattformen und Analysemethoden unterscheiden. Die GPIO- und die Interruptlatenz im Userspace wurden z.B. von Murikipudi et al.[21] für eine ARM Plattform untersucht. Von Garre et al.[9] wurde noch zusätzlich die Scheduling Latenz einer Analyse unterzogen und die Betrachtung ebenfalls auf den Kernelspace ausgeweitet. Verwendet wurde von Garre et al. eine Intel Plattform in Form eines i7-3930K. Reghenzani et al. stellen fest, dass sowohl ein korrekt konfigurierter Kernel als auch Userspaceprogramm nötig sind, um Zuverlässigkeit zu gewährleisten [29][23]. Weiter bemerken sie, dass in der Literatur jedoch häufig die genaue Konfiguration des Systems fehlt, was eine Reproduzierbarkeit erschwert.

Eine der wichtigsten Anpassungen ist es, die CPU-Affinität von Threads und ihrer Interrupt Handler zu setzen, um die Beeinflussung der CPU-Kerne untereinander möglichst gering zu halten. Als am häufigsten untersuchte Metrik wird die Scheduling-Latenz und die Interrupt-Latez angeführt. Diese beiden Metriken sind auch wichtig für diese Arbeit, wohingegen z.B. die Netzwerk-Latenz oder die IPC-Performanz keine Rolle spielen.

Den Einsatz von Xenomai 2 als hartes Echtzeitbetriebssystem in einem Observatoriumsinterferometer auf einer Intel Plattform haben Senata et al. [26] untersucht. Sie verwenden eine Kamera, welche über eine PCIe Karte angeschlossen ist und ihre Daten via DMA transferiert. Diese Kamera ist Teil einer adaptiven Optik zur Verbesserung der Qualität. Sind neue Kameradaten vorhanden, wird von der PCIe Karte ein Interrupt ausgelöst. Als Latenz wurde die Dauer vom Auslösen der Kamera bis hin zum Auftreten des Interrupts gemessen. Es wird lediglich die Gesamtlatenz des Systems angegeben, jedoch nicht die Einzellatenzen wie etwa für den Interrupt. Auch das genaue Vorgehen zur Ermittlung dieser Latenz ist der Arbeit nicht zu entnehmen.

Die Interrupt- und die Schedulinglatenz einer Intel Plattform haben Yang und Shinjo[37] untersucht. Sie verwenden dazu unter anderem als Betriebssysteme Xenomai 3, Preempt_RT und vanilla Linux. Um eine Interruptquelle zu erhalten, benutzen Yang und Shinjo eine serielle Schnittstelle, welche über ein PCIe Gerät bereitgestellt wird. Über diese serielle Schnittstelle generieren sie ein 50% Rechtecksignal mit einer Frequenz von 1-KHz. Die Interruptlatenz wird unter verschiedenen Lastszenarien mit einem Oszilloskop gemessen. Zur Ermittlung der Schedulinglatenz verwenden Yang und Shinjo das Tool Cyclictest.

2.5.2 Cyclictest

Für die Messung von Systemlatenzen unter Echtzeitbetriebssystemen wird häufig das Tool Cyclictest[4] verwendet. Cyclictest startet hierzu eine definierbare Anzahl an parallelen Threads mit einer definierbaren Periode. Zum Ablauf der Periode wechselt der Thread aus dem Sleeping Zustand in den Runnable Zustand. Sobald der Scheduler nun den Thread in den Running Zustand versetzt, wird von diesem ein Zeitstempel genommen und der Thread wechselt zurück in den Sleeping Zustand. Danach wiederholt sich

[4] https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start

der Zyklus. Aus der Differenz der gemessenen Zeit aus dem Zeitstempel und der berechneten geplanten Zeit werden Statistiken über die Latenz, hervorgerufen durch den Software-Interrupt, den Scheduler und die Timerpräzision, erzeugt. Cyclictest kann unter anderem für eine Einschätzung der Worst Case Execution Time verwendet werden. Hierzu muss ein Last-Szenario auf dem System erzeugt werden, welches dem maximalen Lastverhältnis der Plattform und der Anwendung entspricht.

Dennoch gilt es bei der Verwendung von Cyclictest zu beachten, dass die maximale beobachtete Latenz nur eine Einschätzung der Worst Case Execution Time erlaubt. Es kann sein, dass seltene Ereignisse genau zwischen den Intervall Perioden auftreten und somit nicht von Cyclictest erfasst werden[8]. Ebenso merkt das wiki[4] an, dass es kein universell empfehlbares Limit für die gemessene Latenz gibt. Die Messergebnisse müssen daher in Relation zu den Zeitanforderungen für die Echtzeitanwendung interpretiert werden. Weiterhin sind die gemessenen Latenzen „leicht optimistisch“, abhängig von der Echtzeitanwendung, die auf dem System laufen wird.

2.5.3 OSADL - Open Source Automation Development Lab eG

Die Open Source Automation Development Lab eG, kurz OSADL[5], führt in ihrer Echtzeit QA Farm[6] permanente Systembenchmarks mit Cyclictest durch. Ziel von OSADL ist es, die Verwendung von Opensource Software in der Industrie zu stärken und den Entwicklungsaufwand unter den Mitgliedern zu verteilen. Ein Projekt ist besagte Echtzeitfarm, in der mit Hilfe von Stresstests und Systembenchmarks eine Qualtitätssicherung des Preempt_RT Patches stattfindet. Die Plots, die verwendete Hardwareplattform sowie die jeweiligen Kernel-Konfigurationen sind dokumentiert und bilden somit eine Vergleichsgrundlage. Zur Nachvollziehbarkeit der Methoden und damit der Ergebnisse wird der verwendete Cyclictestbefehl nachfolgend aufgeführt:

```
# -m - verhindere das Paging aktueller und
#      zukünftiger Speicheradressierungen
# -Sp90 - Echtzeitpriorität 90
# -l100000000 - Anzahl an Durchläufen
# -i200 - Intervall der periodischen Threads in Mikrosekunden
# -h400 - Erzeuge ein Histogramm und lege
#         die Maximal aufgezeichnete Latenz
```

[5] https://www.osadl.org/
[6] https://www.osadl.org/Latency-plots.latency-plots.0.html

```
#          in Mikrosekunden fest

$ cyclictest -l100000000 -m -Sp90 -i200 -h400
```

Auch stellt OSADL das zum Erzeugen der Plots verwendete Skript unter https://www.osadl.org/uploads/media/mklatencyplot.bash zur Verfügung.

Zum Zeitpunkt des Schreibens dieser Arbeit befinden sich Intel Atom E3845 CPUs, welche auch in der CNC Steuerung verwendet werden, in den Racks d/slot #1 - d/slot #2 - d/slot #3 - und d/slot #4 und b/slot #6. Die Maschinen in den Slots #1-#4 haben identische Motherboards der Firma TQ-Group vom Typ TQMxE38M X64 und als Betriebssystem Debian GNU/Linux 9 (stretch). Slot #6 unterscheidet sich dahingehend, dass ein Motherboard vom Typ TQMxE39M der Firma TQ-Group unter Debian GNU/-Linux 10 (buster) zum Einsatz kommt und die CPU mit 1915 MHz anstelle von 1910 MHz getaktet wird. Die maximal gemessenen Latenzen, sowie die verwendeten Kernel- und Echtzeitpatch-Versionen der Testracks sind in Tabelle 2.1 zu einer Übersicht zusammengefasst. Ein Testdurchlauf entspricht in etwa 5h 30m (100000000 Zyklen * 200µs).

	d/slot #1	d/slot #2	d/slot #3	d/slot #4	b/slot #6
Maximale Latenz	117µs	89µs	98µs	104µs	71µs
Kernel Version	4.19.37-rt19	4.19.37-rt19	4.19.37-rt19	4.19.37-rt19	4.19.37-rt20
Betriebssystem	Debian 9	Debian 9	Debian 9	Debian 9	Debain 10
Taktfrequenz	1910MHz	1910MHz	1910MHz	1910MHz	1915MHz
Board	TQMxE38M X64	TQMxE38M X64	TQMxE38M X64	TQMxE38M X64	TQMxE39M

Tabelle 2.1: OSADL Echtzeit QA Farm Latenzmessungen am 18.10.2021

Diese Werte legen den Erwartungswert fest und dienen als Vergleichsbasis für spätere eigene Messungen.

2.5.4 Perf

Nachdem in Abschnitt 2.5 festgestellt wurde, dass eine exakte Bestimmung der WCET nicht mit realistischem Aufwand erreicht werden kann und somit nur eine Annäherung von unten möglich ist, muss eine Abschätzung der Genauigkeit der gemessenen WCET stattfinden.

Wie bereits unter Abschnitt 2.4 festgestellt wurde, haben die CPU-Caches den größten Einfluss auf die Performanz moderner CPUs. Daher muss die CNC Steuerungssoftware auf ihr Cachingverhalten untersucht werden. Weitere Metriken wie Branch Hits bzw. Misses tragen ebenfalls zu einer verbesserten Einschätzung der gemessenen WCET bei.

„Perf“ ist das offizielle Werkzeug zur Performanzanalyse unter Linux und ist Teil des Linux Kernels unter tools/perf. Perf ist in der Lage, statistische Auswertungen des gesamten Systems zu erstellen, sowohl im Kernel-Space als auch im User-Space. Dabei kann Perf unter anderem auch auf die Hardware Counters (PMCs - performance monitoring counters) der CPU zugreifen [10]. Mit diesen Hardwareregistern können unter anderem folgende Ereignisse beobachtet werden:

- CPU Zyklen
- CPU Instruktionen
- Level 1,2,3 Cache Zugriffe
- Speicher- und Ressourcen I/O

Jede CPU besitzt für gewöhnlich zwei bis acht dieser Hardwareregister, die dazu programmiert werden können, die zuvor aufgeführten Ereignisse aufzuzeichnen. Werden mehr Ereignisse beobachtet als es Register gibt, so wird gemultiplext und das Ergebnis verliert unter Umständen an Genauigkeit, da ein Ereignis nicht mehr die ganze Zeit beobachtet wird [7]. Daher muss, um eine möglichst gute Testabdeckung zu erhalten, die Anzahl der beobachteten Ereignisse mit der Anzahl der Hardwareregister übereinstimmen. Im Falle der in der CNC Steuerung verwendeten CPU sind das zwei Register bzw. Ereignisse [12]. Der Aufbau eines Hardwareregisters für die Intel CPU der CNC Steuerung ist in Abbildung 2.8 dargestellt.

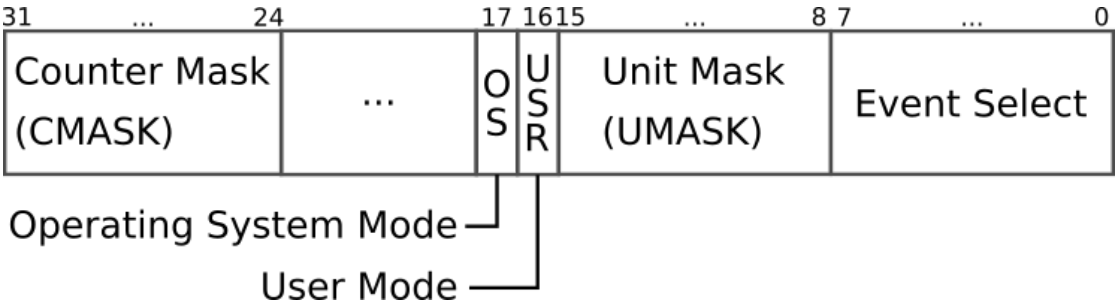


Abbildung 2.8: Aufbau des MSR einer Intel CPU[10]

[7] https://perf.wiki.kernel.org/index.php/Tutorial

Wichtig für die Programmierung sind hierbei die Unitmask und das Event Select. Für die Level2 Cache Misses z.B. ist die UMASK 04H und das Event Select CBH. Mögliche weitere Werte für diese Variablen können [13] entnommen werden. Perf kann mit dem Unterkommando „list“ auch unterstützte Ereignisse anzeigen. So müssen die UMASK und das Event Select nicht manuell eingegeben werden. Mit dem nachfolgenden Befehl lassen sich die Last-Level-Cache-Loads und Misses, also im Falle des Intel Atom E3845 des Level2 Caches, eines Prozesses mit der id „PID“ aufzeichnen.

```
$ perf stat -e LLC-loads,LLC-load-misses -p PID
```

3 Untersuchung Ist-Zustand

Das Gesamtsystem der CNC Steuerung besteht im Wesentlichen aus drei Komponenten, der Hardware, der Steuerungssoftware und dem Betriebssystem. Diese sollen nachfolgend einer Analyse unterzogen werden um die Ausgangssituation festzulegen.

3.1 Hardware

Zu Beginn wird nun die Hardware vorgestellt. In Abbildung 3.1 abgebildet ist die Leiterplattenbohrmaschine, welche die Plattform für die späteren Messungen bietet.

Abbildung 3.1: CNC Leiterplattenbohrmaschine mit sechs Bohrstationen. Modell SD-612 der Firma Tongtai

Es handelt sich um ein Sondermodell der Firma Tongtai [1] mit der Typenbezeichnug SD-612. Sie ist bestückt mit sechs Bohrstationen und kann daher bis zu sechs Leiterplatten gleichzeitig bearbeiten. Als Kernstück der Maschine dient die Bohr-/Frässteuerung CNC 95.00 der Firma Sieb & Meyer. Ihre Komponenten sind in Abbildung 3.2 dargestellt. [34] Sie ist logisch in drei Teile aufgeteilt. Der erste Teil ist das Antriebspaket, bestehend aus 1: PS95, 2: FC95, 3,4: MD95 Nano plus. Das Netzteil PS95 ist die zentrale Spannungsversorgung und das FC95 ist der Frequenzumrichter für den Antrieb der Spindelmotoren. Die beiden Servoverstärker der Serie MD95 Nano plus sind für die Lageregelung und die Positionierung der Maschinenachsen zuständig. Hierbei übernimmt der eine (Nr 3 in Abbildung 3.2) die Antriebe in den X- bzw Y-Achsen und der andere (Nr 4) die Z-Antriebe. Die Servoverstärker gibt es als 1-,2-,4- oder 6-Achsmodule. Sie sind über einen digitalen Bus, SMLink 10, mit dem Frequenzumrichter verbunden.

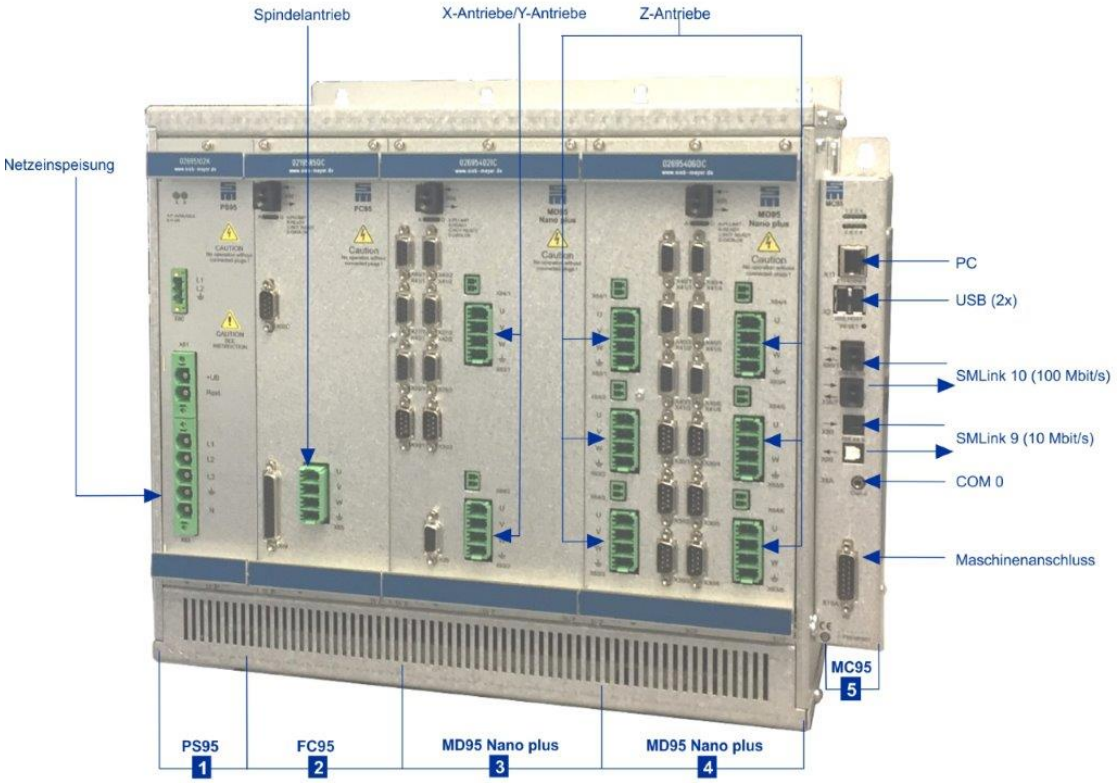


Abbildung 3.2: CNC Steuerung Chassis

[1] https://www.tongtai.com.tw/en/

Der zweite Teil ist der sogenannte Motion Controller 5: MC95. „Der Motion Controller MC95 übernimmt die Positionierung und die interpolierende Bahnplanung in der CNC 95.00. “ [28] und ist via Lichtwellenleiter (SMLink) mit der Maschine verbunden. Über die mit „PC“ betitelte Ethernet-Schnittstelle ist er mit einem weiteren PC, dem Communication Controller, auf dem die CNC Anwendung läuft, verbunden. Dieser bildet den dritten Teil.

Das Board innerhalb des Motion Controllers MC95 ist in Abbildung 3.3 zu sehen. Es beherbergt ein Lattice ECP5 FPGA unter (1) und ein eingebettetes Modul MSC C10M-BTC mit einem Intel Atom E3845, 4GB Hauptspeicher und 4GB eMMC Flash Speicher der Firma Avnet[2] unter (2).

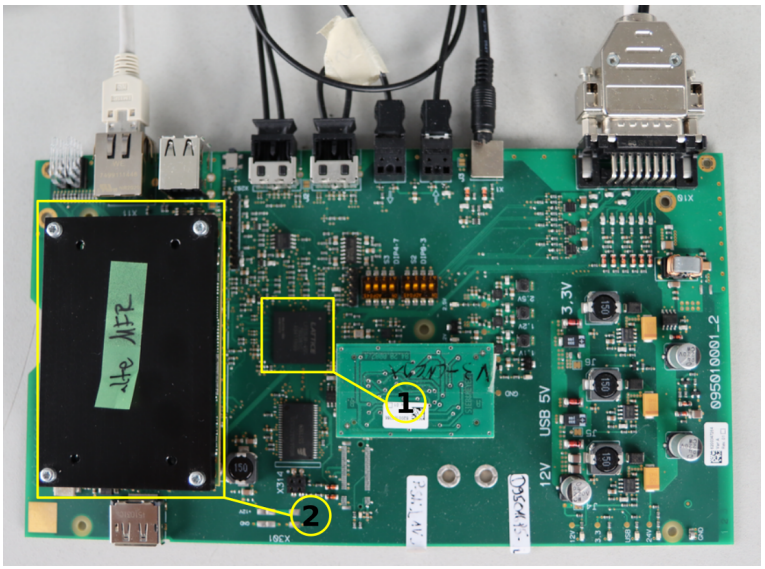


Abbildung 3.3: Motion Controller Hardware - 1: Lattice ECP5 FPGA, angeschlossen über PCIe. 2: Avnet Embedded MSC C10M-BTC unter einem Kühlkörper

Das FPGA erfüllt die Aufgabe eines „Taktgebers“ für das System. Es ist über den PCIExpress Bus mit dem eingebetteten Modul verbunden. Das System ist so konfiguriert, dass die Servomotoren alle 1ms neue Bohr- bzw Fräsdaten vom Motion Controller erwarten. Diese Daten müssen drei Kriterien erfüllen, um akzeptiert zu werden. Sie müssen in einem Takt von 1ms ankommen, gültig sein und es muss sich um neue Daten handeln. Sicherzustellen, dass dieser Takt eingehalten wird, ist von zentraler Bedeutung für diese Arbeit.

[2] https://embedded.avnet.com/product/msc-c10m-btc/

Die Gültigkeit und das Bereitstellen neuer Daten liegen außerhalb der Betrachtung. Erfüllen die Daten eine der drei Anforderungen nicht oder bleiben ganz aus, erhalten die Servomotoren keine neuen Anweisungen und bleiben in diesem Takt entweder stehen oder führen die vorherige Bewegung fort. Dies kann einige Effekte nach sich ziehen. Im besten Fall bleibt die Maschine schlagartig stehen und nimmt die Arbeit wieder auf, sobald wieder korrekte Daten geliefert werden. Der schlimmste Fall könnte eintreten, wenn sich die Maschine im Moment des Ausbleibens neuer Daten in einer Bewegung befindet. Dann besteht eventuell die Gefahr, dass die Bewegung unerwünscht weitergeführt wird und so unter Umständen die Werkstücke und die Maschine selbst Schaden nehmen. Verschiedene Sicherheitsvorkehrungen in der Hardware, auf die hier nicht näher eingegangen wird, garantieren unter Einhaltung der Sicherheitsbestimmungen für die Maschine, dass keine Gefahr für Personen entsteht.

Eine genaue Untersuchung der Steuerungssoftware folgt in Abschnitt 3.2. Um die Rechtzeitigkeit der Daten zu gewährleisten, läuft die CNC Steuerungssoftware auf dem eingebetteten Modul unter einem Echtzeitbetriebssystem. Dieses Echtzeitbetriebssystem wird in Abschnitt 3.3 einer genauen Analyse unterzogen.

3.2 CNC Steuerungssoftware

Im nachfolgenden Abschnitt wird die Software des Motion Controllers untersucht. Zuerst wird der logische Aufbau erläutert und danach wird eine Caching und Branching Analyse der Software mit dem Linux Werkzeug „Perf“ durchgeführt. Abschließend erfolgt eine Betrachtung des zeitlichen Verhaltens der Software.

3.2.1 Logischer Aufbau des Motion Controllers

In Abbildung 3.4 sind die Komponenten der CNC Maschine schematisch dargestellt. Sie besteht im Wesentlichen aus der Maschine, dem Motion Controller und dem Communication Controller. Die Maschine setzt sich wiederum aus Servomotoren und Servoverstärkern sowie Spindeln, einem Frequenzumrichter, einer I/O-Einheit und einem Netzteil zusammen.

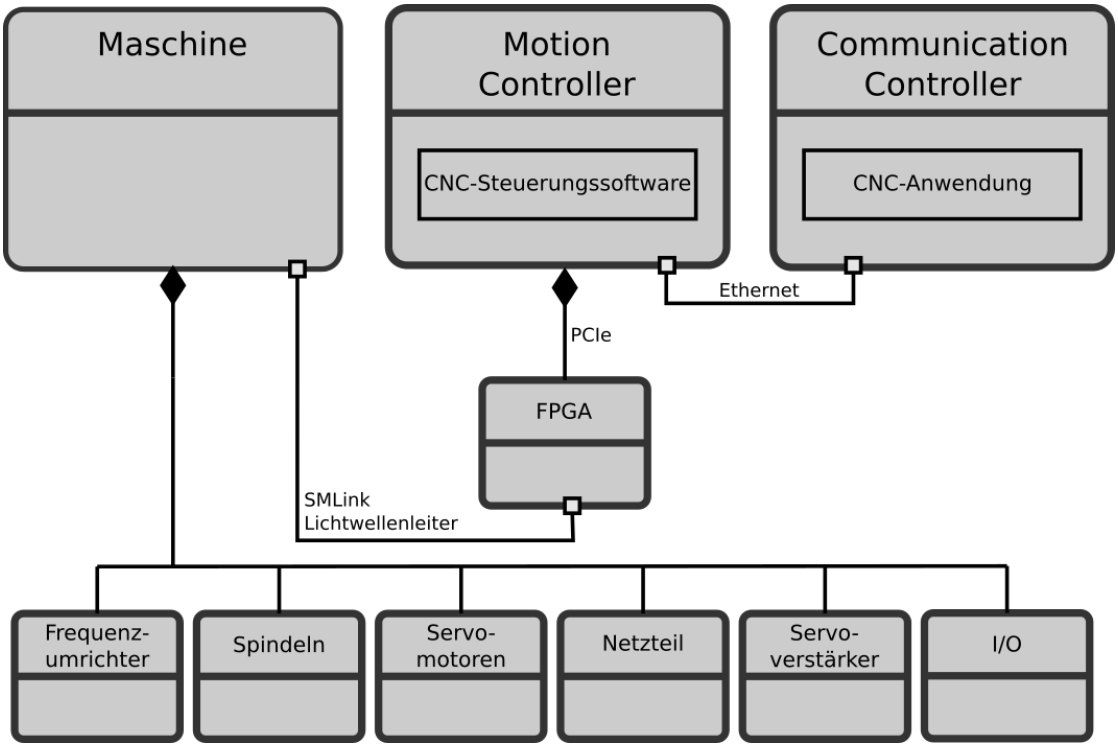


Abbildung 3.4: Architektur der CNC Maschine und des Motion Controllers

Untereinander sind die Komponenten über einen Lichtwellenleiter (SMLink) verbunden. Der Motion Controller ist ebenfalls über einen Lichtwellenleiter an das System angeschlossen. Auf dem Motion Controller läuft die CNC Steuerungssoftware. Sie folgt einem Drei-Schichtenmodell, wie in Abbildung 3.6 zu sehen ist. Die oberste Schicht bildet die Applikationsschicht. Hier wird die Kommunikation mit dem Communication Controller in einem nicht echtzeitkritischen Rahmen durchgeführt. Ebenso findet auch der Datenaustausch über Softwarestrukturen wie Ringpuffer etc. in die darunterliegende Schicht, der logischen Schicht, statt.

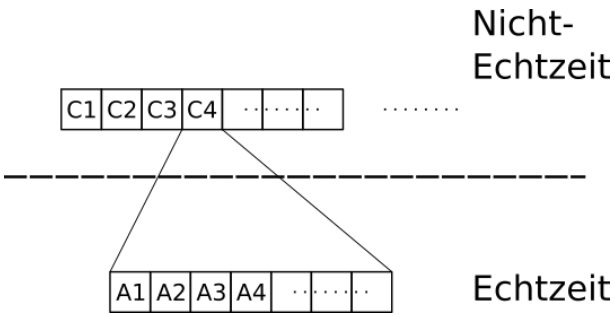


Abbildung 3.5: Übergang von der Nicht-Echtzeit in die Echtzeit

Die logische Schicht markiert den Übergang in den echtzeitkritischen Teil der Anwendung. Hier werden unter anderem die Anweisungen aus dem Bohrprogramm in atomare, also echtzeitkritische, logisch zusammenhängende Bohrdaten umgewandelt und an die unterste Schicht weitergereicht. In Abbildung 3.5 ist dieser Übergang dargestellt. Die Anweisungen des Bohrprogramms in der Darstellung C1 bis Cn, werden in die atomaren Bohranweisungen A1 bis An zerlegt und in einen Puffer im echtzeitrelevanten Teil der Anwendung kopiert.

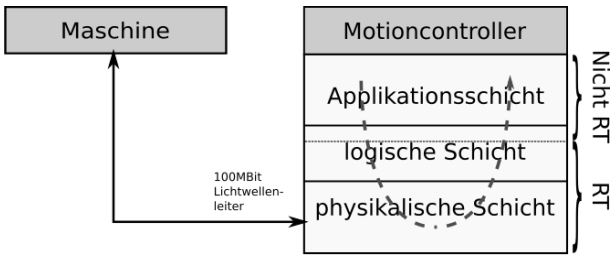


Abbildung 3.6: Schichten der CNC Steuerungssoftware

Im Grunde ist ein atomares Bohrdatum die Summe der Bewegungen in X-, Y- bzw. Z-Richtung, die nötig sind, um ein Loch zu bohren. Während ein atomares Bohrdatum, abgearbeitet wird, darf es zu keiner Unterbrechung kommen und die Daten müssen in dem zuvor erwähnten Takt von 1ms an die Servomotoren gesendet werden. Sind die atomaren Bohrdaten A1 bis An der Anweisung C4 abgearbeitet, so wird die Maschine wieder in einen sicheren Zustand versetzt. So kann sichergestellt werden, dass die Puffer in der Echtzeit durch die Nicht-Echtzeit befüllt werden können. Dieser Takt von 1ms ist in Abbildung 3.7 abgebildet und wird nachfolgend noch genauer erläutert. Die unterste Schicht, die physikalische Schicht, sorgt dafür, dass diese atomaren Bohrdaten in den DMA-Speicher des im Abschnitt zuvor erwähnten FPGAs geschrieben werden, von

wo aus dieses die Daten über den Lichtwellenleiter an die Maschine weiterleitet. Sind alle Einzelinstruktionen des atomaren Bohrdatums abgearbeitet worden, so wird eine Rückmeldung an die oberen Schichten und den Communication Controller gegeben. Dies wird durch den Pfeil in der Abbildung dargestellt. Eine genaue Übersicht über die Architektur der Software im Kontext des Echtzeitbetriebssystems erfolgt in Abschnitt 3.3 in Abbildung 3.14b.

Der zuvor erwähnte Takt von 1ms ist in Abbildung 3.7 dargestellt und wird nach einer Setup-Phase vom FPGA erzeugt. Zu Beginn des Zyklus holt das FPGA Bohrdaten aus dem DMA-Speicher. Dieses kopiert seinerseits Statusdaten in den DMA-Speicher. Sind die Kopiervorgänge im DMA-Speicher abgeschlossen, wird vom FPGA ein Interrupt generiert, welcher den eigentlichen Beginn des Zyklus markiert. Die Motion Controller Software liest nun die Statusdaten aus dem DMA-Speicher und berechnet daraus die Bohrdaten für die nächste Bohrung. Befindet sich der Zyklus noch unterhalb der Dauer von 500µs, so werden evtl noch optionale Berechnungen oder andere hier nicht weiter spezifizierte Daten verarbeitet.

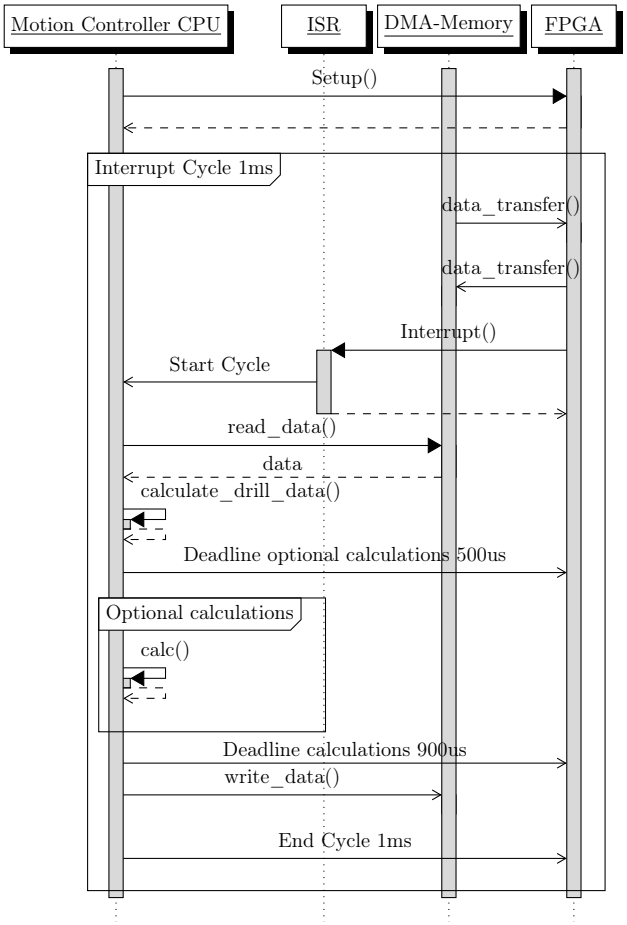


Abbildung 3.7: Sequenzdiagramm des Interruptzyklus der CNC Steuerung

Alle Berechnungen müssen bis zur Marke von 900µs abgeschlossen sein. Danach ist ein Zeitraum von 100µs reserviert, in dem die Ergebnisse der Berechnungen von der Motion Controller Software in den DMA-Speicher kopiert werden. Diese werden dann im nächsten Zyklus wieder vom FPGA gelesen und der Zyklus beginnt erneut.

3.2.2 Kategorisierung des Systems

Im Folgenden soll nun das System anhand der Kriterien aus Abschnitt 2.1 einem der Echtzeitbegriffe zugeordnet werden.

Wie schnell muss das System reagieren können? Aus dem Sequenzdiagramm aus Abbildung 3.7 ist ersichtlich, dass es alle 1ms eine zyklische Deadline gibt. Innerhalb dieser 1ms sind am Ende 100µs für Datentransfer über den DMA-Speicher reserviert. Somit muss das System in der Lage sein alle nötigen Berechnungen innerhalb von 900µs abzuschließen.

Wie streng sind die Deadlines? Das System toleriert ein Verpassen der Deadline um 300µs für maximal drei aufeinanderfolgende Zyklen.

Wie zuverlässig muss das System sein? Das System muss sehr zuverlässig sein, da die CNC Steuerung hohen Qualitätsansprüchen genügen muss und bei Verfehlung die Werkstücke unbrauchbar werden können, was sowohl einen zeitlichen als auch einen materiellen Verlust bedeutet. Bei Einhaltung aller Sicherheitsvorschriften besteht jedoch keine Gefahr für Personen.

Was ist der Umfang des Systems und was ist der Grad der Interaktion unter den Komponenten? Das System ist umfangreich. Bohrdaten können im Hauptspeicher Gigabyte Größen erreichen. Die Interaktion zwischen den echtzeitkritischen Komponenten wurde so gering wie möglich gehalten.

Wie verhält sich die Umgebung, in der das System läuft? Die Umgebung, in der das System läuft ist, dynamisch. Die physikalische Beschaffenheit der Werkstücke wie etwa die Dicke (µm Bereich) ist zuvor nicht bekannt. Auch ist das Inputprogramm des Benutzers dynamisch.

Ausgehend von den Charakteristika der CNC Steuerung und ihrer Anforderungen lässt sich das System am besten als „wahrscheinliche harte Echtzeit“ einordnen. Es ist wünschenswert dass alle Deadlines eingehalten werden, um die Bohr- und Fräsgeschwindigkeit aufrecht zu erhalten und eine hohe Qualität zu sichern. Sollte jedoch in sehr seltenen Fällen eine Deadline verpasst werden, so zieht das noch keine katastrophalen Folgen nach sich.

3.2.3 Analyse des zeitlichen Verhaltens

Nachfolgend wird das zeitliche Verhalten der CNC Steuerungssoftware einer Analyse unterzogen. Auf Bitten von Sieb & Meyer wird an dieser Stelle auf eine exakte Analyse des Laufzeitverhaltens mit einem Scatter Plot verzichtet.

Logisch ist die Software in sechs sequentielle Abschnitte aufgeteilt. Diese Abschnitte bilden den Zyklus aus Abbildung 3.7 ab und werden im weiteren Verlauf mit A bis F bezeichnet. In den Abschnitten A und B wird auf den Interrupt des FPGAs reagiert und der Datentransfer aus dem DMA-Speicher an die Servomotoren durchgeführt. Der Abschnitt C ist für die Berechnung der Bohrdaten und für die Datenverarbeitung im Takt zuständig. Abschnitt D enthält keine Berechnungen und markiert lediglich das Ende der logischen Schicht. Die Abschnitte A bis D sind echtzeitrelevant, wohingegen E und F optionale Bereiche darstellen, die nur dann ausgeführt werden, wenn A bis D zusammen unter 500µs gedauert haben. In E werden sicherheitsrelevante Daten verarbeitet und Informationen in regelmäßigen Abständen an den Communication Controller weitergereicht. Abschnitt F beinhaltet optionale Berechnungen für den nächsten Bohrhub wie z.B. die Optimierung der Z-Achsen Bewegung.

Die CNC Steuerungssoftware bietet die Möglichkeit, Zeitstempel zu Beginn der sechs Abschnitte zu nehmen. Für die folgenden CNC Anwendungsfälle wurden diese Zeitstempel aufgezeichnet und daraus die Laufzeiten für die einzelnen Abschnitte berechnet. Dabei wurden die Bohr- und Fräsprogramme so gewählt, dass sie zum einen häufig vorkommende Anwendungsfälle beinhalten und zum anderen erwartungsgemäß eine möglichst große Last erzeugen. Dadurch wird die maximale Ausführungszeit ermittelt. 6z bzw. 2z bezeichnet jeweils die Anzahl an aktiven Bohrstationen, also Z-Achsen.

Bohren cbd jlimit 6z Abgearbeitet wird ein Bohrprogramm mit aktiviertem cbd (contact board drilling). Hier werden Fehler simuliert und in optionalen Berechnungen ein optimiertes Bewegungsmuster des Bohrkopfes erstellt, wodurch eine erhöhte Bewegungsgeschwindigkeit erreicht werden kann, wenn sich der Bohrkopf in der Luft befindet. Durch jlimit (jerk limit) findet eine Ruckbegrenzung bei der Bewegung statt. Es wird ein weicheres Bewegungsprofil erstellt, welches im Gegensatz zum regulären Bewegungsprofil erhöhten Rechenaufwand verursacht. Die Profilplanung findet in den optionalen Bereichen E und F statt. Verwendet werden sechs Bohrstationen.

Bohren zspd jlimit 6z Entspricht dem Bohrprogramm von zuvor, jedoch ohne cbd. Dafür wurde die Optimierung zspd aktiviert. Dadurch werden die Bewegungen des Bohrkopfes in X-,Y- und Z-Richtung gleichzeitig ausgeführt. Verwendet werden sechs Bohrstationen.

Standard Bohrtestrogramm 2z Das Standard Bohrtestprogramm von Sieb & Meyer. Verwendet werden zwei Bohrstationen.

Standard Bohrtestprogramm 6z Das Standard Bohrtestprogramm von Sieb & Meyer. Verwendet werden sechs Bohrstationen.

Tiefenbohren 2z Ein Bohrprogramm, bei dem besonders tiefe Löcher gebohrt werden. Verwendet werden zwei Bohrstationen.

Tiefenbohren 6z Ein Bohrprogramm, bei dem besonders tiefe Löcher gebohrt werden. Verwendet werden sechs Bohrstationen.

Peck 6z Bei Pecking wird eine Bohrung in Teilbohrungen mit zunehmender Bohrtiefe aufgeteilt. Dadurch gibt es eine Neuberechnung der Bohrung bei jedem Kontakt des Bohrers mit der Oberfläche. Dies verursacht einen erhöhten Rechenaufwand und lässt sich nicht im Voraus planen. Verwendet werden sechs Bohrstationen.

Fräsen Es werden verschiedene Formen gefräst. Zum Zeitpunkt der Untersuchung des zeitlichen Verhaltens befanden sich Teile der Fräsfunktion der CNC Steuerung noch in der Entwicklung, wodurch Optimierungen fehlen und dadurch höhere Laufzeiten auftreten können.

Leerlauf Die Maschine befindet sich im Leerlauf, es wird kein Bohrprogramm abgearbeitet. Dies dient der Ermittlung der Grundlast auf dem System, welche durch Funktionen des Betriebssystems oder auch Statusmeldungen der CNC Steuerung an den Communication Controller verursacht wird.

Werkzeugwechsel Es wird ein Werkzeugwechsel durchgeführt. Hier müssen Werkzeugvermessungsvorgänge abgearbeitet werden ebenso wie eine Reihe an Sonderskripten, wie etwa für die personelle Sicherheit.

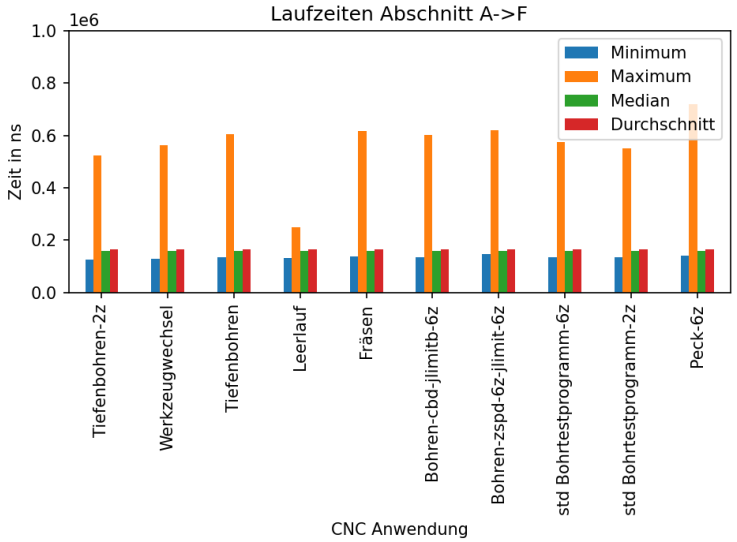


Abbildung 3.8: Minimum, Maximum, Durchschnitt und Median des gesamten Zyklus unter verschiedenen Anwendungsfällen

Abbildung 3.8 zeigt die minimal und maximal gemessene Ausführungszeit. Ebenso ist der Durchschnittswert und der Median zu sehen. Zu beobachten ist, dass der Durchschnittswert und der Median bei einer Ausführungszeit von unter 200µs für alle Anwendungsfälle nahe am minimalen Wert liegen. Im Leerlauf beträgt der Maximalwert 250µs. Für die anderen Anwendungsfälle liegt er zwischen 522µs beim Tiefenbohren mit zwei Z-Achsen und 721µs bei Peck. Größtenteils sind die Ausführungszeiten also recht niedrig mit sporadisch auftretenden hohen Spitzen. Der höchste gemessene Wert sind die zuvor erwähnten 721µs bei Peck.

Werden die Gesamtlaufzeiten in die einzelnen Abschnitte A bis F aufgeschlüsselt, so ergibt sich das Bild aus Abbildung 3.9. Verantwortlich für die maximale Laufzeit in A ist der Anwendungsfall Werkzeugwechsel. Für das Maximum in B ist es Fräsen, für C Peck, für D wieder Fräsen, für E der Werkzeugwechsel und für F Bohren zspd jlimit 6z. Verantwortlich für das aufaddierte Maximum von A bis D, A bis E und A bis F ist in allen drei Fällen Peck. Dieser Anwendungsfall wird später noch genauer betrachtet.

Die Graphen der restlichen Anwendungsfälle befinden sich der Vollständigkeit halber im Anhang.

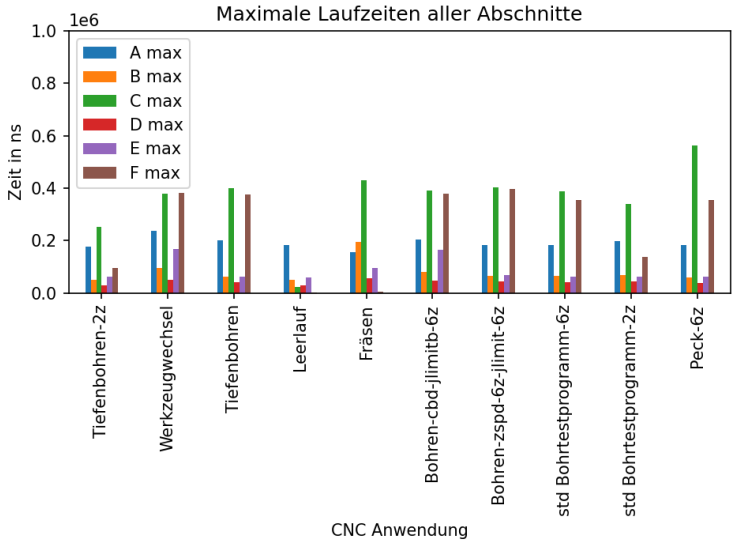


Abbildung 3.9: Maximale Laufzeiten der Softwareabschnitte A bis F der CNC Steuerung unter verschiedenen Anwendungsfällen

In der Darstellung ist ebenfalls zu sehen, dass die optionalen Abschnitte E und F – abgesehen vom Leerlauf – noch beim Tiefenbohren mit 2z, beim Fräsen und beim Standard Bohrtestprogramm mit 2z nicht, oder nur sehr wenig verwendet werden. die Maxima liegen hier deutlich unter 200µs. Der Abschnitt A liegt bei allen Anwendungsfällen in etwa um 200µs und der Abschnitt B bei ca. 60µs. Einzige Ausnahme für den Abschnitt B bildet das Fräsen, wo der Wert bei 196µs liegt. Dies lässt sich dadurch erklären, dass durch fehlende Optimierungen größere Mengen an Daten per DMA kopiert und gelesen werden müssen.

Für ein potentielles Maximum dürfen die Maxima der einzelnen Abschnitte jedoch nicht einfach aufaddiert werden. Die Abschnitte E und F stellen, wie zu Beginn erwähnt, optionale Berechnungen dar, die nur dann stattfinden, wenn die Abschnitte A bis einschließlich D zusammen unter 500µs liegen. Daher werden die Maxima für E und F nur dann auf die Gesamtsumme addiert, wenn A bis D unter 500µs liegen. Ansonsten werden nur die Minima von E und F hinzu addiert. Das Ergebnis dieses Aufaddierens ist in Abbildung 3.10 zu sehen. Die Addierten Abschnitte A bis E und A bis F sind gleich hoch, mit der Ausnahme für den Leerlauf. Dies bedeutet, dass für die Berechnung der potentiellen Maxima

die Werte für die Abschnitte A bis D immer über den 500µs lagen und somit keine optionalen Berechnungen durchgeführt wurden. Das Maximum liegt bei 930µs sowohl beim Fräsen als auch beim Werkzeugwechsel. Peck liegt mit 903µs knapp dahinter.

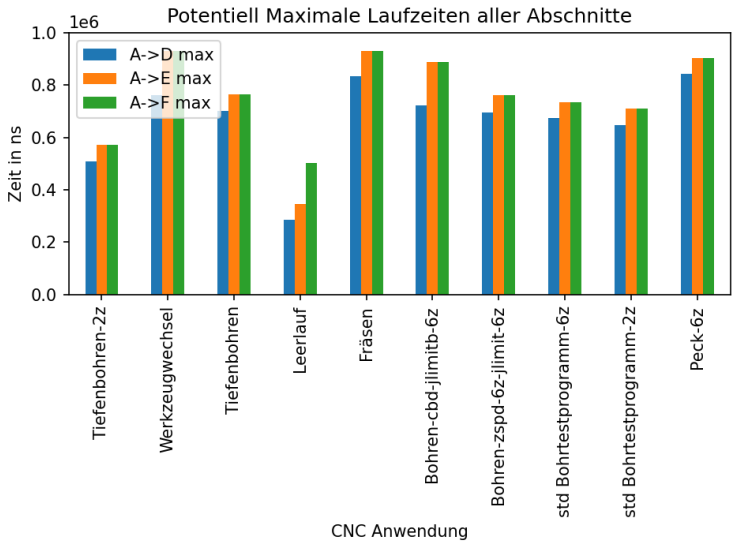


Abbildung 3.10: Potentielles Maximum der Softwareabschnitte A bis F der CNC Steuerung unter verschiedenen Anwendungsfällen

Die potentiellen Maxima liegen mit der Ausnahme des Leerlaufs und des Tiefenbohrens mit 2z bei 800µs und höher. Bezieht man die Deadline bei 900µs für alle Berechnungen ein, so ist bei vielen Anwendungsfällen kaum Luft nach oben und in sehr seltenen Fällen besteht beim Fräsen und bei Peck die theoretische Möglichkeit eine Deadline zu verpassen. Diese Information fliest bereits in die Entwicklung der Steuerungssoftware ein und es wird versucht, die recht niedrige Grundlast von Durchschnittlich unter 200µs zu erhöhen und dafür die Spitzen abzuflachen.

Exemplarisch wird der Anwendungsfall Peck, welcher für die tatsächlich gemessene maximale Laufzeit verantwortlich ist, noch näher betrachtet. Zu Beginn kann festgestellt werden, dass die Abschnitte B und D mit maximal 60µs und 37µs kaum relevant für das Maximum von 720µs sind. Der Bereich A und B sind wie bereits erwähnt unter den Anwendungsfällen recht ähnlich, da hier nur der DMA-Transfer stattfindet. Es ist nicht überraschend, dass die größte Spitze im Bereich C mit 563µs liegt, da bei Peck bei jeder Teilbohrung erneut die Bohrdaten berechnet werden müssen.

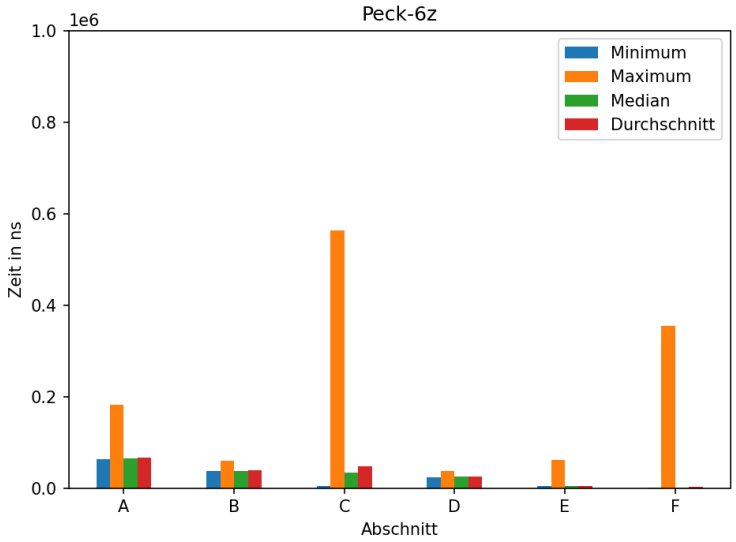


Abbildung 3.11: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Peck

Für diesen Maximalwert in C werden keine optionalen Berechnungen in E und F durchgeführt. Selbst C liegt im Durchschnitt bei lediglich 47µs. Werden optionale Berechnungen angestellt, so liegt hier das Maximum der Laufzeit in F bei 354µs.

3.2.4 Caching und Branching Analyse

Die tatsächlich aufgetretenen maximalen Laufzeiten verschiedener Anwendungsfälle wurden ermittelt. Es wurde ebenfalls aus den einzelnen Laufzeiten der logischen Abschnitte ein hypothetisches Maximum für die Laufzeit errechnet. Eine exakte Bestimmung der WCET ist in diesem Fall nicht realistisch, wie unter Kapitel 2 festgestellt wurde. Dennoch führt eine Betrachtung des Caching- und Branching-Verhaltens der Software zu einer besseren Einschätzung der Laufzeiten. Profitiert die Software sehr von den Caches und werden selten die falschen Branches vorausberechnet, so gibt es an dieser Stelle kaum Verbesserungsmöglichkeiten für die Laufzeiten und die Maxima verschlechtern sich tendenziell. In diesem Fall ist bei Änderungen der Software besonders darauf zu achten, wie sich die Laufzeiten verhalten.

	Fräsen		Bohren	
	1z	6z	1z	6z
L1-icache-load-misses in %	6,33	6,44	9,22	9,22
LLC-load-misses in %	2,71	2,82	2,31	2,32
LLC-store-misses in %	8,65	9,63	7,22	7,34
Branch-misses in %	6,78	6,85	8,49	8,46

Tabelle 3.1: Cache und Branch Misses der Software beim Bohren und Fräsen

Ist die Ausnutzung der Caches und die Vorhersage der Branches eher schlechter, so sind die ermittelten Maxima recht solide zu bewerten.

Mit perf wurde für die Anwendungsfälle Fräsen und das Standard Bohrprogramm mit jeweils einer und sechs aktiven Bohrstationen das Caching- und Branching-Verhalten aufgezeichnet.

Ermittelt wurden die L1 Instruction Cache Load Misses und für den L2 Cache die Load und Store Misses. Ebenfalls gemessen wurde die Anzahl der falsch vorausberechneten Branches. Zunächst eine Erklärung der Begrifflichkeiten [11]:

L1-icache-load-misses Diese Metrik zeigt an, wie oft der Versuch etwas aus dem Level 1 Instruktions Cache zu laden, fehlgeschlagen ist.

LLC-load-misses Diese Metrik zeigt an, wie oft der Versuch etwas aus dem Last Level Cache zu laden, fehlgeschlagen ist und daher ein Zugriff auf den Hauptspeicher nötig war. Der Last Level Cache ist im Falle des Intel Atom E3815 der Level 2 Cache.

LLC-store-misses Durch diese Metrik wird angegeben, wie oft zuerst die Cache Line in den LLC gelesen werden musste, bevor der LLC Schreibzugriff abgeschlossen werden konnte. Dies geschieht durch zuvor aufgetretene unvollständige Schreibzugriffe und dadurch inkonsistente Daten.

Branch-misses Diese Metrik sagt aus, wie oft von der CPU falsch vorhergesagt wurde welcher Branch einer if-then-else Struktur genommen werden muss. Durch einen Miss werden die Instruktionen in der Pipeline der CPU ungültig und die Pipeline muss erst neu befüllt werden, was zu einer Verringerung der Ausführungsgeschwindigkeit führt.

Ein Ermitteln der Hits bzw. Misses für die L1 Daten Caches war nicht möglich, da dies von der CPU nicht unterstützt wird. Die Ergebnisse sind in Tabelle 3.1 zusammengefasst. Auffällig ist, dass es keinen nennenswerten Unterschied zu machen scheint, ob mit einer oder mit sechs Z-Achsen gebohrt bzw. gefräst wird. Einzige Ausnahme bildet das Fräsen in Bezug auf die Last Level Cache Store Misses, wo in der sechs Z-Achsen Variante 1% mehr Misses aufgetreten sind. Insgesamt 9,63% im Vergleich zu 8,65% für eine Z-Achse. LLC Store Misses betragen rund 7% beim Fräsen. Die L1 iCache Load Misses betragen rund 6% beim Fräsen und rund 9% beim Bohren. Eine mögliche Erklärung für den Unterschied von etwa 3% ist, dass beim Fräsen andere Bewegungsprofile verwendet werden und die Frästiefe meist oder zumindest für längere Zeit konstant bleibt. Interessant sind die mit ca. 2,3% bzw 2,4% für Bohren und Fräsen gleichermaßen sehr niedrigen LLC Cache Misses. Dies ist der wichtigste gemessene Wert, da bei Misses im LLC ein Zugriff auf den im Verhältnis sehr langsamen RAM stattfinden muss. Die CNC Anwendung hat mit unter 2,5% Misses im LLC eine sehr gute Ausnutzung der Daten Caches. Folglich muss die ermittelte maximale Laufzeit konservativ betrachtet und mit etwas „Spielraum nach oben“ gerechnet werden.

3.3 Xenomai 2

Im nachfolgenden Abschnitt wird das Echtzeitbetriebssystem der CNC Steuerung genauer betrachtet. Zunächst werden die Architektur der Anwendung und des Betriebssystems untersucht. Eine entscheidende Komponente hierbei ist ein Gerätetreiber, der in Funktion und Aufgabe danach detailliert vorgestellt wird. Zuletzt werden die Echtzeiteigenschaften betrachtet, indem die Scheduling Latenz und die Interrupt Latenz ermittelt werden.

3.3.1 Architektur der CNC Steuerung unter Xenomai 2

Das Betriebssystem des Motion Controllers basiert auf GNU+Linux mit dem Entwicklungsframework Xenomai[3], welches den Linuxkernel um harte Echtzeitfähigkeiten er-

[3] https://source.denx.de/Xenomai/xenomai/-/wikis/home

weitert. Xenomai wurde 2002 von Philippe Gerum veröffentlicht und orientiert sich im Aufbau an der „Adaptive Domain Environment for Operating Systems“ von Karim Yaghmour[35]. Xenomai folgt einem sogenannten Dualkernel Ansatz dargestellt in Abbildung 3.12. OS-Domain 1 entspricht dem Xenomai-Echtzeitkernel und OS-Domain 2 dem Linuxkernel.

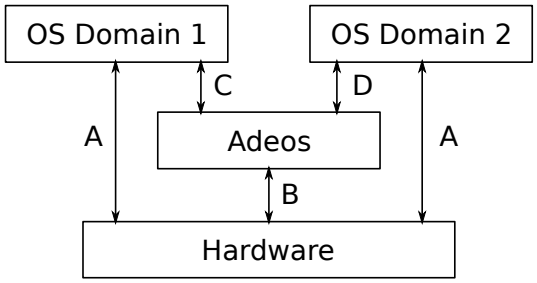


Abbildung 3.12: Architektur Adeos[35]

Adeos stellt vier Methoden zur Kommunikation bereit. Die erste Methode, dargestellt mit A in der Abbildung, bezeichnet allgemeine Nutzung der Hardware. Dies beinhaltet unter anderem Speicherzugriffe und Zugriffe auf Hardwarekomponenten. Methode B kommt zum Einsatz, wenn Adeos durch Software- bzw. Hardwareinterrupts die Kontrolle erhält. Einbezogen sind hier auch alle Hardwarebefehle die durch ADEOS ausgeführt wurden, um die Hardware zu steuern. Kategorie C beinhaltet das Ausführen einer Interruptroutine des Betriebssystems. Der Interruptkontext wird hierbei durch ADEOS bereitgestellt. Als letztes besteht noch die Methode D, die eine Zwei-Wege-Kommunikation zwischen einer Domain und ADEOS beschreibt. Dies kann genutzt werden, um Zugriff auf die Ressourcen oder die komplette Kontrolle einer anderen Domain anzufragen. Möglich ist dies nur für Domains, die von der Existenz des ADEOS wissen. Im Falle von Xenomai kann über diesen Weg der Xenomai-Echtzeitkernel völlige Kontrolle über den Linuxkernel erlangen. Um Echtzeit gewährleisten zu können, werden Interrupts von ADEOS in einer Interrupt-Pipeline nach ihrer Echtzeitbedeutung an den entsprechenden Kernel weitergereicht. Die Interrupt-Pipeline oder kurz Ipipe ist in Abbildung 3.13 zu sehen. In der Theorie können beliebig viele Domains in der Pipeline vorhanden sein. Bei Xenomai sind dies jedoch nur zwei, wie oben beschrieben. Der Linuxkernel verarbeitet demnach nur Interrupts, die nicht echtzeitrelevant sind und wenn ihm Rechenzeit vom Xenomaikernel zugeteilt wird. Das Betriebssystem besteht also aus drei Komponenten: dem Linux Kernel, dem Xenomai-Framework und dem ADEOS/Ipipe-Patch. In der CNC Steuerung kommt Xenomai in der

Version 2.6.4 und der Linuxkernel inklusive IPipe-Patch in Version 3.14.17 unter 32Bit zum Einsatz.

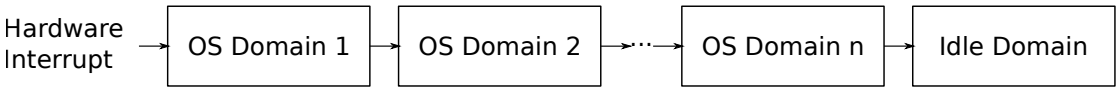
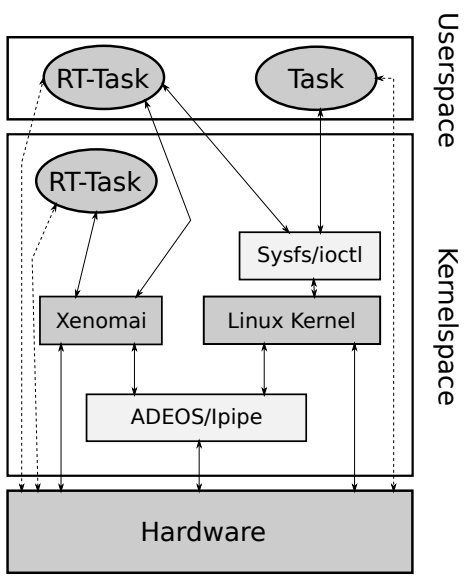
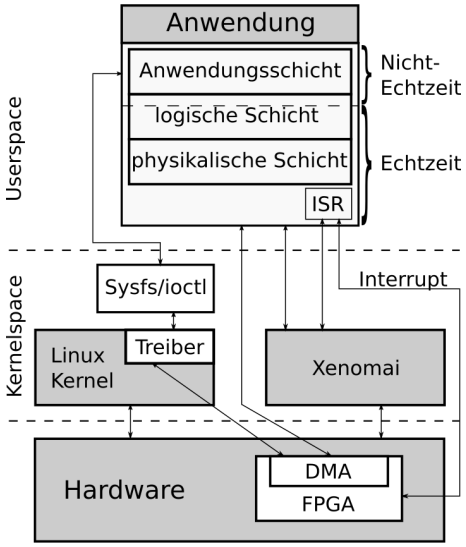


Abbildung 3.13: Interrupt pipeline von Adeos[35]

Ein Echtzeit-Task kann entweder im Kernelspace oder im Userspace laufen, zu sehen in Abbildung 3.14a. Ebenfalls zeigt das Bild, wie sich der Dualkernel Ansatz in die Gesamtarchitektur eingliedert. Die CNC Steuerungssoftware läuft im Userspace und kommuniziert über das Linux Sysfs bzw. das ioctl-Interface mit einem Kerneltreiber. Die Aufgabe dieses Linux Kernel Treibers ist es zum einen besagte Kommunikationsschnittstelle für die CNC Steuerungssoftware bereitzustellen und zum anderen ein Interface zum FPGA in Form eines DMA-Speichermappings zu gewährleisten.



(a) Architektur Xenomai



(b) Architektur der CNC Steuerungssoftware unter Xenomai 2

Abbildung 3.14: Architektur unter Xenomai 2

Der Treiber ist eine wichtige Komponente der CNC Steuerung und wird in Unterabschnitt 3.3.2 noch einmal ausführlich betrachtet. Eine vollständige Übersicht der Architektur zeigt Abbildung 3.4. Die Interrupt Service Routine (ISR) für den taktgebenden Interrupt des FPGAs ist nicht im Treiber implementiert, sondern als Teil der CNC

Steuerungssoftware im Userspace. Xenomai 2 erlaubt, es eine ISR direkt im Userpace zu implementieren.

3.3.2 Gerätetreiber

Das Xenomai 2 Framework bietet eine API für ein „Real-Time Driver Model“ (RTDM) für Treiber, die Echtzeitfunktionalitäten benötigen. Dies ist jedoch nicht der Fall für den Treiber der CNC Steuerung. Der Treiber verwendet Standard Linux Interfaces und lediglich die Schnittstellen in Form von ioctl, um das FPGA zu konfigurieren und zu initialisieren. Das RTDM muss nicht umgesetzt werden, da die echtzeitkritische ISR für den Interrupt, welcher durch das FPGA erzeugt wird, direkt durch das Xenomai 2 Interface in der CNC Steuerungssoftware implementiert ist[4].

Das FPGA ist über eine PCIExpress Schnittstelle angebunden. Gemäß der PCIExpress Basis Spezifikation [22] besitzt das PCIExpress Gerät sechs Basis Adress Register (BAR). Das FPGA wird über ein DMA-Subsystem konfiguriert, welches diese BARs in den Hauptspeicher mappt. Dazu muss, wie in den Grundlagen beschrieben, der Kernel zuerst die Busadressen der BARs in den physikalischen Adressraum der CPU mappen. Danach mappt der Treiber die physikalischen Adressen in den virtuellen Adressraum des Kernels. Um diesen Adressraum aus dem Userspace zu erreichen, muss noch in einem weiteren Speichermapping der Adressraum in den Userspace gemappt werden.

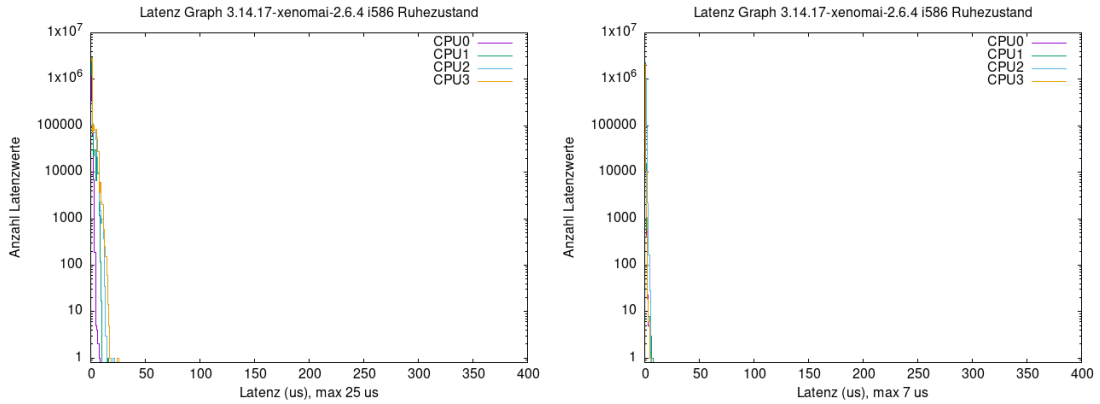
Durch die Eigenheit des Xenomai 2 wird die Verarbeitung der Interrupts innerhalb der CNC Steuerungssoftware durchgeführt und nicht im Treiber. Außer der Konfiguration des FPGA und der Bereitstellung der DMA-Adressen erfüllt der Treiber somit keine weitere Funktion.

3.3.3 Performanzmessung mit Cyclictest

Für eine erste Einschätzung der Performanz des verwendeten Betriebssystems wird die Scheduling Latenz mit Cyclictest gemessen. Die Parameter sind aufgrund der besseren Vergleichbarkeit denen von OSADL angeglichen, daher erhält die X-Achse eine recht große Breite von 400µs. In einem ersten Versuch, zu sehen in Abbildung 3.15a, wurde über einen Zeitraum von ca. sechs Stunden eine maximale Latenz von 25µs unter Idle-Bedingungen gemessen. Bei der Betrachtung des Graphen fällt auf, dass die vier Kurven

[4] https://www.xenomai.org/documentation/xenomai-2.6/html/api/

alle hintereinander stattzufinden scheinen. Zu erwarten wäre, dass sich bei parallelen Prozessen die Kurven wenigstens etwas überschneiden. Dies legt nahe, dass die Cyclictest Version des Xenomai 2 die Threads nicht auf die vorhandenen CPUs verteilt und daher sequentiell auf einem Kern abarbeitet.



(a) Idle 6h Versuch 1

(b) Idle 6h Versuch 2

Abbildung 3.15: Cyclictest Messungen Xenomai2 32Bit Ruhezustand

In einem zweiten Anlauf wurde Cyclictest viermal angestartet, jedoch jeweils mit nur einem Thread und mit dem Tool „taskset“ auf die vier Kerne der CPU verteilt. Anschließend wurden aus den vier erzeugten Einzelausgaben wieder eine gemeinsame Tabelle für alle vier Threads erzeugt. Das Ergebnis dieses Versuches ist in Abbildung 3.15b dargestellt. Dieses Mal konnte eine Latenz von 7µs über einen Testzeitraum von ca. sechs Stunden unter Idle-Bedingungen ermittelt werden. Zur Nachvollziehbarkeit der Ergebnisse sind die verwendeten Parameter in Listing 3.1 aufgeführt.

```
# 1. Versuch
cyclictest -t4 -p99 -n -i200 -l100000000 -d0 -q

# 2. Versuch
taskset -c0-3 cyclictest -t1 -p99 -n -i200 -l100000000 -d0 -q
```

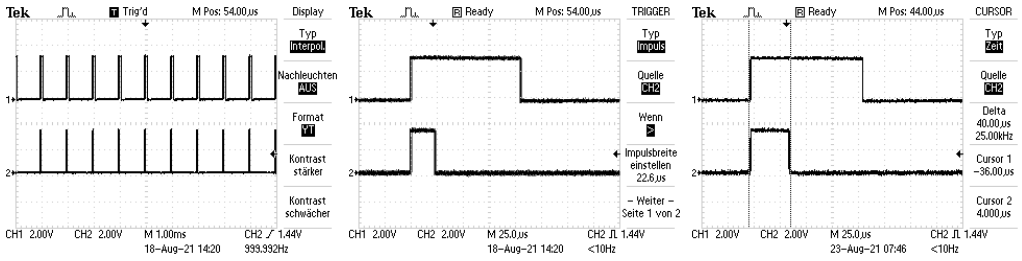
Listing 3.1: Verwendete Paramter für Cyclictest für Versuch 1 und 2

Insgesamt mussten mehrere Anläufe für die Versuche gestartet werden, da das Cyclictest-Tool des Xenomai 2 häufige Abstürze verursachte und unzuverlässige Ausgaben produzierte. Im weiteren Verlauf wird daher bei Xenomai 2 auf Cyclictest verzichtet.

3.3.4 Performanzmessung mit Oszilloskop

Nachdem zuvor mit Cyclictest eine Einschätzung der zu erwartenden Scheduling Latenz ermittelt wurde, soll nun die Interrupt Latenz des Systems gemessen werden. Um das System sich nicht selbst testen zu lassen und um eine möglichst große Objektivität zu erreichen, kommt ein Oszilloskop zum Einsatz. Für die Messung wird das Oszilloskop TDS 1012 von Tektronix [32] verwendet.

Das System bietet hierzu vier programmierbare Pins, welche über das FPGA angesprochen werden können. Ein Pin wurde so konfiguriert, dass er für ca. 100µs auf High gezogen wird, sobald von dem FPGA ein Interrupt generiert wird. Dieser Pin dient der Überprüfung, ob das FPGA vom Treiber richtig initialisiert wurde und es in der Lage ist, Interrupts zu generieren. Um die realen Gegebenheiten der CNC Steuerung nachzubilden, wurde das FPGA so programmiert, dass es in einem Takt von 1ms Interrupts erzeugt. In Abbildung 3.16a auf Kanal 1 ist dieses Signal zu sehen.



(a) Interrupt Test (b) Pulsweiten Test (c) Last >96h

Abbildung 3.16: Oszilloskop Messungen Xenomai2 32Bit Last

Damit die Interrupt Latenz direkt vom Oszilloskop abgelesen und ein Pulsweiten-Trigger verwendet werden kann, wird ein zweiter Pin verwendet. Dieser Pin ist so konfiguriert, dass er ebenso wie der erste Pin auf High gesetzt wird, sobald ein Interrupt vom FPGA erzeugt wurde. Zurückgesetzt wird der Pin jedoch nicht nach einer festen Zeit, sondern in einem Thread, welcher von der ISR erzeugt wird. Die Zeit, in der der zweite Pin auf High steht, entspricht demnach der Interrupt Latenz und wird auf Kanal 2 dargestellt. In Abbildung 3.16a ist dieses Signal ebenfalls in einem Abstand von 1ms zu sehen.

Demnach ist das FPGA richtig konfiguriert und es werden alle 1ms Interrupts generiert. Es kann nun also ein Pulsweiten-Trigger auf dem zweiten Kanal gesetzt werden, um zu sehen, ob eine bestimmte Breite nicht überschritten wird. Um sicherzustellen, dass

der Pulsweiten-Trigger korrekt auslöst, wurde zuerst in Abbildung 3.16b die Pulsweiten soweit reduziert, bis der Trigger auslöst. Dies war, wie ebenfalls im Bild zu sehen, bei 22,6µs der Fall. Für den eigentlichen Test wurde ein möglichst langer Zeitraum von 96 Stunden gewählt und der Pulsweiten-Trigger auf größer gleich 39µs gestellt. Zusätzlich wurde ein Lastszenario für den Test gewählt, welches der maximal zu erwarteten Last im Regelbetrieb der CNC Steuerung entspricht. Hierzu wurde auf dem Steuerungs-PC das Tool „yes“, welches einen String kontinuierlich auf der Konsole ausgibt und so 100% CPU Auslastung verursacht, einmal pro CPU-Kern ausgeführt. Um eine hohe Auslastung des Hauptspeichers zu simulieren, wurde eine RAM-Disk erzeugt, in die mit dem Befehl

```
$ openssl rand -out /tmp/random -base64 $(( 2**30 * 3/4 ))
```

1GB an Zufallsdaten in einer Endlosschleife geschrieben wurden. Um schließlich auch noch die Netzwerkschnittstelle auszulasten und eine große Anzahl an Interrupts auf dem Testsystem zu erzeugen, wurde über einen weiteren PC ein Flood-Ping mit

```
$ ping -s 65000 -f IP-TARGETPC
```

durchgeführt. Somit wurden in einer Dauerschleife Zufallsdaten mit der Größe 1GB mit dem Befehl „scp“ über das Netzwerk in die RAM-Disk des Testsystems geschrieben. Ebenso wurde die Funktion „Nachleuchten“ des Oszilloskops aktiviert und auf *unendlich* gestellt, um sehen zu können, wie häufig und in welchem Maße die Marke von 39µs überschritten wird. In dem gesamten Testzeitraum betrug die maximal gemessene Interrupt Latenz lediglich 40µs, wie in Abbildung 3.16c zu sehen ist.

4 Auswahl Zielsystem

An dieser Stelle sei noch einmal die Zielsetzung dieser Arbeit aufgeführt. Es sollen mögliche Nachfolger für das derzeitig verwendete Echtzeitbetriebssystem Xenomai 2 der CNC Steuerung CNC95.00 gesucht werden und der Portierungsaufwand eingeschätzt werden. Dazu wurde in den vorhergehenden Kapiteln die Architektur der CNC Steuerung und ihre Laufzeitumgebung untersucht, um nun daraus die Anforderungen an diesen möglichen Nachfolger abzuleiten.

In diesem Kapitel soll die Auswahl des Zielsystems geschehen. Zuerst werden die Anforderungen an das Echtzeitbetriebssystem ermittelt. Danach wird systematisch untersucht, welche der zur Verfügung stehenden Echtzeitbetriebssysteme diese Anforderungen erfüllen. Zuletzt wird der Portierungsaufwand für die ausgewählten Optionen beleuchtet.

4.1 Anforderung an das Echtzeitbetriebssystem

Aus der vorhandenen Laufzeitumgebung und der Architektur der CNC Steuerung lassen sich Anforderungen an das Echtzeitbetriebssystem ableiten. Unterschieden wird hier zwischen zwei Kategorien von Anforderungen. Die erste Kategorie beinhaltet alle absolut notwendigen Anforderungen und Vorgaben durch Sieb & Meyer. Auch solche Punkte, die einen unrealistischen Portierungsaufwand bedeuten, sind Teil dieser Kategorie. Die zweite Kategorie bilden wünschenswerte aber dennoch optionale Anforderungen.

Folgende Punkte müssen unbedingt erfüllt werden:

Unterstützung für Intel x86 Architektur Die CNC Steuerung läuft unter einer Intel Plattform.

Ethernet und TCP/IP Der Motion Controller erhält die Bohrdaten über einen weiteren PC via Ethernet und meldet seinen Status ebenfalls über Ethernet.

CIFS/NFS Ein Dateiaustausch mit dem Communication Controller findet über ein via Ethernet gemountetes CIFS Laufwerk statt. Eine Umstellung dieser Kommunikation bedeutet zusätzlichen Portierungsaufwand.

PCIExpress/DMA Das FPGA, welches den Takt für das System gibt, benötigt Support für PCIExpress und DMA. Treiber für beides zu schreiben, bedeutet einen unrealistischen Portierungsaufwand.

Multithreading Die CNC Steuerung führt sehr rechenintensive echtzeitrelevante Berechnungen Parallel aus. Ebenso laufen viele nicht-echtzeitrelevante Dienste, die während dieser Berechnungen verdrängt werden würden und z.B. das Feedback an den Benutzer stark verzögern.

Frei verfügbar/nicht proprietär Vorgabe durch Sieb & Meyer.

sehr gute Debugmöglichkeiten Ein wichtiger Teil der Entwicklung ist das Finden von Fehlern. Tools wie strace und gdb sind essentiell. Ebenfalls ein ausdrücklicher Wunsch von Sieb & Meyer.

SSH Eine SSH Verbindung ermöglicht leichtere Fehlerdiagnose, Wartung und Debugging.

Die weiteren Punkte sind wünschenswert, jedoch keine K.O.-Kriterien:

32Bit Die CNC Steuerungssoftware wurde auf 32Bit ausgelegt. Ein Umstieg auf 64Bit bedeutet eine Verdopplung der Zeigergrößen und somit einen Verlust an Hauptspeicher ebenso wie einen erhöhten Portierungsaufwand.

Zukunftssicher Stellt eigentlich ein wichtiges Kriterium dar, jedoch lässt sich die Zukunft eines Projektes schwer vorhersagen.

Unterstützung für weitere Plattformen Ähnlich dem Punkt zuvor. Wünschenswert, falls eine zukünftige Plattform nicht mehr Intel-basiert ist.

4.2 Mögliche Kandidaten für das Echtzeitbetriebssystem

Zuerst sollen mögliche Kandidaten für das Echtzeitbetriebssystem ermittelt werden. Eine Recherche auf der Internetseite osrtos[1] und Wikipedia[2] ergibt eine große Anzahl an Optionen, die nachfolgend aufgeführt sind (Stand Juni 2021):

ATK2, AliOS Things, Apache Mynewt, Atomthreads, Azure RTOS, BRTOS, BeRTOS, BitThunder, ChibiOS/RT, Contiki OS, Contiki-NG, Drone, DuinOS, Erika Enterprise, F9 Microkernel, Femto OS, FreeRTOS, Freescale MQX, Frosted, FunkOS, Fusion Embedded RTOS, HyperC, IntrOS, LibreRTOS, LiteOS, MARK3, MOE, MaRTE, Mongoose OS, Nut/OS, NuttX, Phoenix-RTOS, Preempt_RT, Prex, Protothreads, QuarkTS, RIOT, RT-Thread, RTAI, RTEMS, StateOS, StratifyOS, TI-RTOS Kernel, TNKernel, TNeo, TencentOS-tiny, TinyOS, TizenRT, Tock, Trampoline, Xenomai, Zephyr, cocoOS, distortos, eChronos, eCos, embox, mbed OS, scmRTOS, seL4, uC/OS-II, uC/OS-III, uKOS, uSmartX

Um eine Struktur in diese Optionen zu bekommen, werden zuerst alle nicht frei verfügbaren Betriebssysteme entfernt und alle, die keine x86-Plattform unterstützen. Ebenso entfernt werden alle Projekte, deren Status inaktiv ist oder die seit mehr als einem Jahr keine Updates erhalten haben. Dies verkürzt die Liste, abgebildet in Tabelle 4.1 beträchtlich auf neun mögliche Kandidaten.

Name	Lizenz	zuletzt Upgedated
FreeRTOS	MIT	2021-04-29
NuttX	Apache License 2.0	2021-07-18
Phoenix-RTOS	BSD	2021-07-15
Preempt_RT	GPL v2	2021-03-19
QuarkTS	LGPL	2021-06-14
RT-Thread	Modified GPL v2	2021-05-28
Xenomai 3	GPL	2021-07-22
Zephyr	Apache License	2021-06-04
seL4	GPL v2	2021-06-10

Tabelle 4.1: Echtzeitbetriebssystem Kandidaten nach Entfernen aller proprietären Optionen und jener, welche keine x86-Plattform unterstützen.

[1] https://www.osrtos.com/
[2] https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

Im Folgenden werden diese neun Betriebssysteme anhand der zuvor festgelegten Anforderungen überprüft. Es werden hierbei Noten von 1 bis 3 vergeben, wobei 1 bedeutet, dass das Betriebssystem mit nur geringem bis mittlerem Portierungsaufwand verwendbar ist. Note 2 bedeutet mit Aufwand nutzbar und mit Note 3 werden die für die CNC Steuerung schlichtweg ungeeigneten Kandidaten bewertet.

FreeRTOS Ist ein Echtzeitbetriebssystem von Amazon Web Services und wird daher von einer großen, namhaften Firma entwickelt. Die Ausrichtung ist eher auf Mikrocontroller. Es wird DMA unterstützt, jedoch weder SSH, noch CIFS/NFS, noch PCIExpress. Note 3.

NuttX Ist ein Echtzeitbetriebssystem der Apache Software Foundation. Auch hier ist die Ausrichtung eher auf Mikrocontroller. CIFS/NFS wird unterstützt, ebenso wie PCI-Express und DMA, jedoch kein SSH. Trotz der fehlenden SSH Unterstützung ist NuttX einen Blick wert, sollte es keine besser geeigneten Alternativen geben. Note 2.

Phoenix-RTOS Ist ein Echtzeitbetriebssystem mit Mikrokernel mit Fokus auf IoT Geräte. Es fehlt die Unterstützung für DMA, PCIExpress, CIFS/NFS und SSH. Note 3.

Preempt_RT Linux mit dem Preemt_RT Patch ist ein volles Betriebssystem mit exzellenter Hardwareunterstützung und einem großen Angebot an Software. Es erfüllt alle Anforderungen. Der Portierungsaufwand wird später in diesem Kapitel untersucht. Ebenso werden die Echtzeiteigenschaften des Betriebssystems zu einem späteren Zeitpunkt ermittelt. Note 1.

QuarkTS Ist ausgerichtet auf kleine eingebettete Anwendungen und ist nicht Voll Preemtive. Es hat keine Unterstützung für SSH, PCIExpress oder CIFS/NFS. Somit ist es ungeeignet für die CNC Steuerung. Note 3.

RT-Thread Ist ein Echtzeitbetriebssystem mit Ausrichtung auf das IoT. Es bietet ein POSIX Interface, welches ermöglicht Linux/Unix Programm nach RT-Thread zu portieren. DMA, sowie CIFS/NFS werden unterstützt, jedoch fehlt die Unterstützung für PCIExpress und SSH. Note 3.

Xenomai 3 Als linuxbasiertes Betriebssystem und da es die Nachfolgerversion des eingesetzten Betriebssystems Xenomai 2 ist, erfüllt es alle Anforderungen. Xenomai 3 ist daher der Hauptkandidat für einen Umzug der CNC Steuerungssoftware. Im späteren Verlauf wird der Portierungsaufwand genauer betrachtet und die Echtzeiteigenschaften des Betriebssystems untersucht. Note 1.

Zephyr wird von der Linux Foundation für das IoT entwickelt. Es unterstützt PCIExpress mit MSI-X und DMA, jedoch kein SSH und kein NFS. Zephyr fällt damit in die zweite Kategorie und ist einen Blick wert, wenn es an Alternativen mangelt. Note 2.

seL4 Ist eigentlich ein Betriebssystem Mikrokernel und besitzt daher nur einen kleinen Teil davon, was heute als Betriebssystem angesehen wird. PCIExpress und DMA werden unterstützt, genauso wie TCP/IP über Ethernet. Der Status der Bibliothek für CIFS/NFS ist jedoch inaktiv und es konnte keine Unterstützung für SSH verifiziert werden. Dadurch wird der Portierungsaufwand zu hoch und seL4 bleibt lediglich eine Option, sollte es keine anderen Kandidaten geben. Note 2.

Linux mit dem Preempt_RT-Patch und Xenomai 3 können demnach als primäre Kandidaten für weitere Untersuchungen identifiziert werden. Sollte sich im Laufe der Untersuchungen herausstellen dass beide Optionen in Bezug auf die Echtzeiteigenschaften nicht für die CNC Steuerung geeignet sind, so stehen mit NuttX, Zephyr und seL4 weitere Kandidaten zur Verfügung — wenn auch mit deutlich erhöhtem Portierungsaufwand.

4.3 Portierungsaufwand

An dieser Stelle ergibt sich ein Problem. Zur Erinnerung: Teil der CNC Steuerung ist ein Linux Treiber, welcher die Schnittstelle für die Konfiguration des über PCIExpress angeschlossenen FPGAs bereitstellt. Um später die Interruptlatenz der möglichen Kandidaten für das Echtzeitbetriebssystem messen zu können, muss das FPGA programmiert

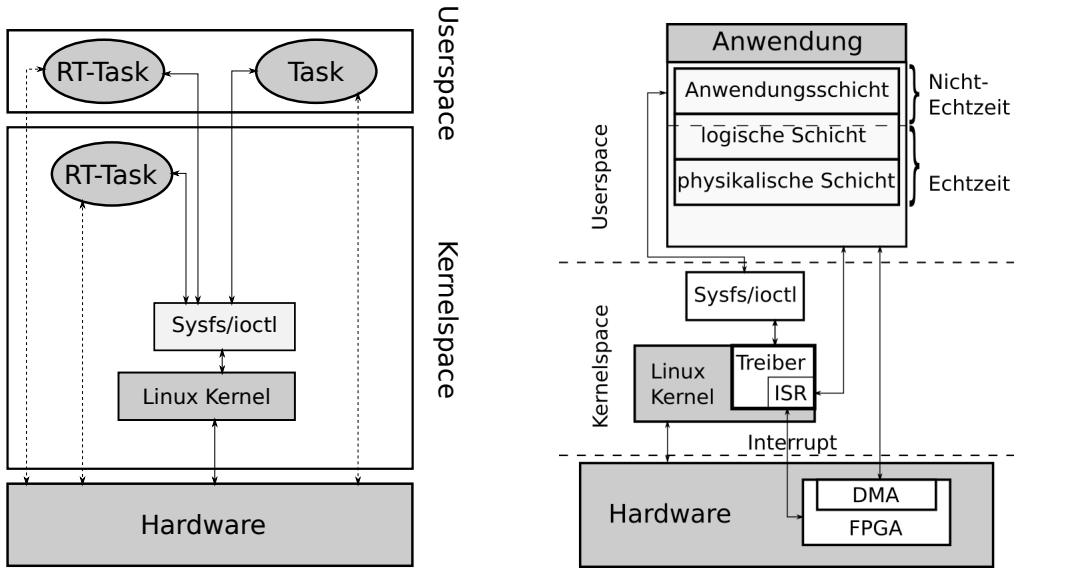
werden. Bevor also eine Einschätzung der Echtzeiteigenschaften eines Systems erfolgen kann, muss der Linux Treiber auf diese Plattform portiert werden. Der Portierungsaufwand muss demnach bereits erbracht werden, bevor feststeht, ob das System den Echtzeitanforderungen der CNC Steuerung genügt.

Für nicht Linux-basierte Ansätze ist dieser Portierungsaufwand ungleich höher. Zum einen wegen besagtem Linux Treiber, zum anderen auch wegen der fehlenden nötigen Anforderungen. Hier müsste entweder die CNC Steuerungssoftware in Bezug auf den Dateiaustausch über CIFS/NFS geändert werden, sollte dies nicht verfügbar sein, oder eine Unterstützung müsste implementiert werden. Analog gilt dies für die anderen Anforderungen.

Nachfolgend wird untersucht, wie sich der Portierungsaufwand für die beiden primären Kandidaten verhält.

4.3.1 Architektur Preempt_RT

Zunächst soll betrachtet werden, wie sich die Architektur der CNC Steuerung unter Preempt_RT im Vergleich zu Xenomai 2 verändert. Die Ausgangsarchitektur zeigt Abbildung 3.14 auf Seite 46. Die Architekturänderung ist in Abbildung 4.1 zu sehen.



(a) Architektur Preempt_RT (b) Architektur MC unter Preempt_RT

Abbildung 4.1: Architektur unter Preempt_RT

Auf der linken Seite in Abbildung 4.1b ist die allgemeine Architektur unter Linux mit Preempt_RT Patch abgebildet. Es gibt hier nur einen Single Kernel, der den Zugriff auf die Hardware regelt und über das sysfs, bzw ioctl Interface Funktionalitäten für den Userspace bereitstellt. Echtzeittasks können sowohl im Kernelspace als auch im Userspace laufen.

Die rechte Seite in Abbildung 4.1b zeigt wie sich die CNC Steuerungssoftware in diese Architektur einfügt. Ein bedeutender Unterschied zu der Architektur unter Xenomai 2 ist es, dass es nicht mehr möglich ist, die ISR im Userspace zu implementieren. Unter Preempt_RT ist es nötig, die ISR in den Linux Treiber und damit in den Kernelspace zu verschieben. Dadurch entsteht eine „Barriere“ zwischen der ISR und der CNC Steuerungssoftware. Nämlich jene zwischen Userspace und Kernelspace. Tritt ein Interrupt auf, wird dieser nun im Kernelspace im Treiber, genauer in der ISR im Treiber entgegengenommen und quittiert. Jetzt muss noch die Steuerungssoftware vom Eintreffen dieses Interrupts durch den Treiber informiert werden, damit diese einen neuen Zyklus starten kann. Im nachfolgenden Abschnitt wird die nötige Änderung für den Treiber und der Mechanismus zum Überbrücken dieser Barriere und Informieren über den Interrupt erläutert.

Die CNC Software verwendet Threads und Timer der Xenomai API. Als weitere Folge bei einem Umstieg auf Preempt_RT müssen diese durch native Linux Aufrufe ersetzt werden. Der Portierungsaufwand an dieser Stelle ist jedoch als gering einzuschätzen.

4.3.2 Linux Gerätetreiber unter Preempt_RT

Durch den Versionssprung von 3.14 auf 5.4 muss der Treiber an die API des neuen Kernels angepasst werden. Ein Versuch den Treiber unter der neuen Kernel Version zu kompilieren, identifiziert die entsprechenden Änderungen und die Dokumentation für die 5.4 Version[3] erläutert die neue API. Aufgrund der Architekturänderung unter Preempt_RT ergeben sich weitere nötige Änderungen für den Linux Treiber. Durch das Wegfallen der Xenomai 2 API kann die ISR des PCIExpress Gerätes nicht mehr als Teil der CNC Steuerungssoftware implementiert werden. Somit wandert die ISR aus der Anwendung und dem Userspace in den Treiber und somit den Kernelspace. Wie im Abschnitt zuvor erläutert, ergibt sich dadurch eine Barriere zwischen der ISR und der Steuerungssoftware. Damit die Anwendung möglichst ohne Zeitverlust über das Auftreten eines Interrupts informiert werden kann, müssen im Treiber „Simple Waiting Queues“ implementiert werden. Diese

[3] https://www.kernel.org/doc/html/v5.4/

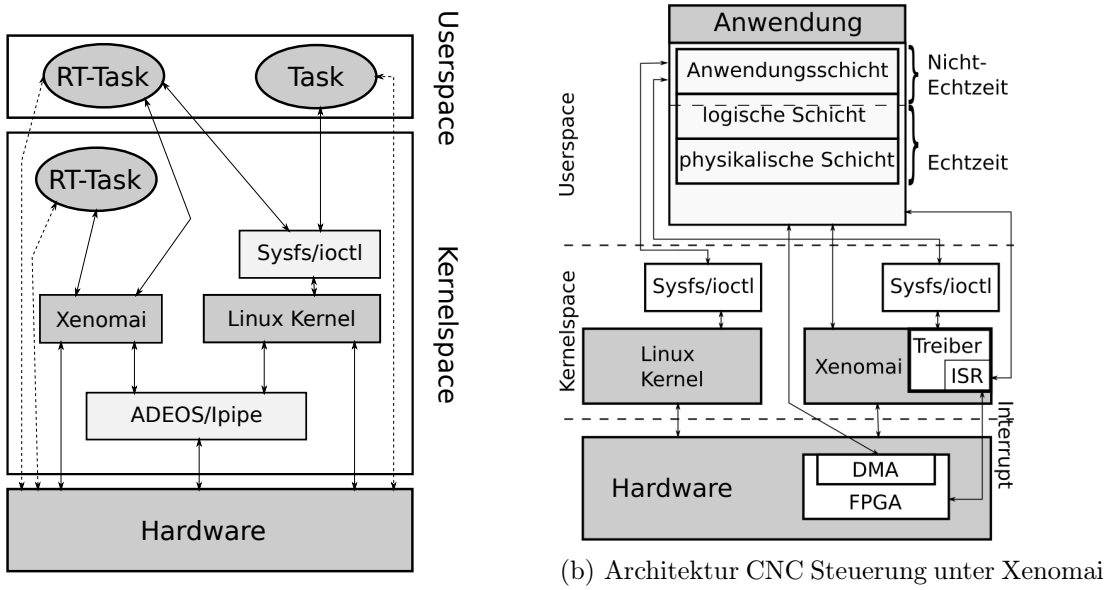
SWQs wurden für Echtzeitanwendungsfälle entwickelt und werden von Jonathan Corbet auf lwn.net[4] genau beschrieben. Sie werden dafür verwendet, um Threads zu verwalten, die darauf warten, dass ein Ereignis eintritt. Dieses Ereignis ist im Falle der CNC Steuerungssoftware der Interrupt durch das FPGA. Das Einreihen des Userspace Threads in diese Simple Waiting Queue geschieht über das ioctl Interface des Treibers. Dazu startet die Anwendung in einer Schleife ein blockierendes Lesen auf den vom Treiber erzeugten Geräteknoten. Wird nun ein Interrupt im Treiber registriert, so wird der Thread der Anwendung in der SWQ aufgeweckt.

Nach wie vor ist der Treiber unter Preempt_RT für die Konfiguration und Initialisieren des FPGAs zuständig. Hinzugekommen ist nun die Aufgabe des Registrierens und Verarbeitens der vom FPGA erzeugten Interrupts. Im Treiber ebenso hinzugekommen ist die ioctl Schnittstelle, um besagte Interrupts an die CNC Steuerungssoftware im Userspace zu melden.

4.3.3 Architektur Xenomai 3

Die Änderungen, die sich durch einen Umstieg auf Xenomai 3 ergeben, werden nachfolgend betrachtet. Die grundlegende Architektur ist bei Xenomai 3 mit Dualkernel identisch mit der des Xenomai 2, wie Abbildung 4.2a zeigt. Dennoch ergeben sich im Detail Unterschiede, wie die rechte Seite unter Abbildung 4.2b nahelegt. Xenomai 3 erlaubt es nicht mehr eine Interrupt Service Routine im Userspace zu implementieren. Somit wandert die ISR aus der CNC Software in den Gerätetreiber. Dadurch ergeben sich zwei Probleme: Zum einen muss, wie schon im Abschnitt zu Preempt_RT beschrieben, die Barriere zwischen Userspace und Kernelspace überwunden werden. Zum anderen läuft der gesamte Treiber im Linux Kernel und nicht im Xenomai Cokernel, was dazu führt, dass die ISR keine Echtzeitpriorität mehr besitzt. Um den Treiber in den Bereich des Xenomai Cokernels zu bekommen, muss der Treiber auf die API des „Real-Time Driver Models“ (RTDM) umgestellt werden. Diese API muss unter Xenomai für Treiber verwendet werden, welche Echtzeitfunktionalitäten benötigen.

[4] https://lwn.net/Articles/577370/



(a) Architektur Xenomai 3

(b) Architektur CNC Steuerung unter Xenomai 3

Abbildung 4.2: Architektur unter Xenomai3

Unter Xenomai 2 war die Verwendung des RTDM nicht nötig, da der Treiber keine Echtzeitrelevanten Aufgaben erfüllen musste. Die ISR konnte unter Verwendung der Xenomai API als Teil des Userspace Programms implementiert werden. Der Treiber musste lediglich das Mapping des DMA-Speichers in den Kernelspace vornehmen und die Adressen für den Userspace zugänglich machen. Somit konnte das FPGA über den DMA-Speicher ohne weitere Interaktion mit dem Treiber oder dem Linux Kernel konfiguriert werden.

Als Portierungsaufwand muss in der CNC Software die ISR entfernt und im Treiber implementiert werden. Für Threads und Timer kann weiterhin die Xenomai API verwendet werden. An dieser Stelle gibt es keinen weiteren Portierungsaufwand.

Nachfolgend werden die nötigen Änderungen in Bezug auf den Treiber näher erläutert.

4.3.4 Linux Gerätetreiber unter Xenomai 3

Wie bereits unter Preempt_RT muss der Treiber auch unter Xenomai 3 an die API der neuen Kernel Version 5.4 angepasst werden. Jedoch ergeben sich auch unter Xenomai 3 Architekturänderungen und damit verbunden, nötige Anpassungen am Gerätetreiber. Xenomai 3 entfernt die Möglichkeit, die ISR im Userspace zu implementieren. Somit

muss analog zu Preempt_RT die ISR in den Treiber wandern. Dies ist aber noch nicht ausreichend, da der Treiber im Linux Kernel läuft und nicht im Xenomai Cokernel. Ein auftretender Interrupt des FPGAs würde also durch den ipipe-Patch als niedriger priorisiert eingestuft werden und unterliegt somit keinem Echtzeitkontext. Der Treiber und damit die ISR müssen demnach im Kontext des Xenomai Cockernels laufen. Dazu muss das RTDM[5] im Treiber implementiert werden. Ebenso wie unter Preempt_RT zuvor ergibt sich das Problem, dass die Anwendung im Userspace von dem Auftreten des Interrupts in der ISR des Treibers benachrichtigt werden muss. Dazu bietet das RTDM eigene Wait Queues, die ebenso wie unter Preempt_RT funktionieren. Die CNC Anwendung muss auch hier in einer Schleife blockierend von dem Treiberknoten lesen. Dadurch reiht sie sich in die Wait Queue ein und wird bei einem auftretenden Interrupt von dem Treiber aufgeweckt.

4.3.5 32Bit vs. 64Bit

Als zusätzlichen Portierungsaufwand hat sich der im Falle von Xemomai 3 nötige Umstieg auf 64Bit herausgestellt. Unter Xenomai 3 wurde für die x86 Plattform der Support für 32Bit eingestellt. Für den letzten unterstützten Linux Kernel konnte die Version 4.9.146 identifiziert werden.

Aktuelle und zukünftige Xenomai Versionen erfordern demnach zwingend eine Portierung auf 64Bit. Der Vorteil ist offensichtlich: Es kann mehr als 4GB Hauptspeicher adressiert werden. Die für die CNC Steuerung verwendete Hardware besitzt jedoch nur 4GB, wodurch dieser Vorteil potentiell erst für zukünftige Hardware zum Tragen kommt. Für das aktuelle Board bedeutet ein Umstieg zunächst eine Verdopplung des Speicherbedarfs für Pointer. Da die internen Datenstrukturen der CNC Anwendungen massiven Gebrauch von Pointern machen, führt dies möglicherweise zu einem merklichen Verlust an zur Verfügung stehendem Hauptspeicher. Bei einer möglichen Portierung ist darauf zu achten, in welchem Umfang sich dieser Mehrverbrauch an RAM bewegt.

Für den Linux Kernel, auch mit Preempt_RT Patch, existiert eine Alternative zu einem „harten Umstieg“ auf 64Bit. Es besteht die Möglichkeit einen 64Bit Kernel zu verwenden und 32Bit Userspace Programme über eine Kompatibilitätsschicht laufen zu lassen. Dazu muss der Treiber, welcher hier unter 64Bit läuft, wie in der Kernel Dokumentation[6]

[5] https://www.xenomai.org/documentation/xenomai-3/html/xeno3prm/group__rtdm__driver__interface.html

[6] https://www.kernel.org/doc/html/latest/driver-api/ioctl.html

beschrieben, eine zweite Variante des ioctl callback Handlers für 32Bit Zugriffe aus dem Userspace implementieren. Wenn über diese ioctl Schnittstelle Datenstrukturen ausgetauscht werden sollen, wie dies auch der Fall ist bei der CNC Steuerung, ist darauf zu achten, dass keine Typen mit variabler Länge wie long und unsigned long verwendet werden. Ebenso muss auf das Padding der Daten geachtet werden, da unter 64Bit Systemen Daten auf 64Bit aligned sind im Gegensatz zu dem alignment von 32Bit unter 32Bit Systemen. Wird dies nicht beachtet, so umfasst die Datenstruktur im Kernelspace evtl. mehr Bytes als im Userspace, was dazu führt, dass auf falsche Speicheradressen zugegriffen wird.

Für Xenomai 3 ist dies jedoch keine Option, da hier das RTDM, welches aus den in den vorhergehenden Abschnitten aufgeführten Gründen verwendet werden muss, keine API für die 32Bit-kompatiblen ioctl callback Handler besitzt. Im Falle von Xenomai 3 führt demnach kein Weg an 64Bit und dem damit verbundenen Portierungsaufwand vorbei.

5 Performanzmessung von Echtzeitbetriebssystemen

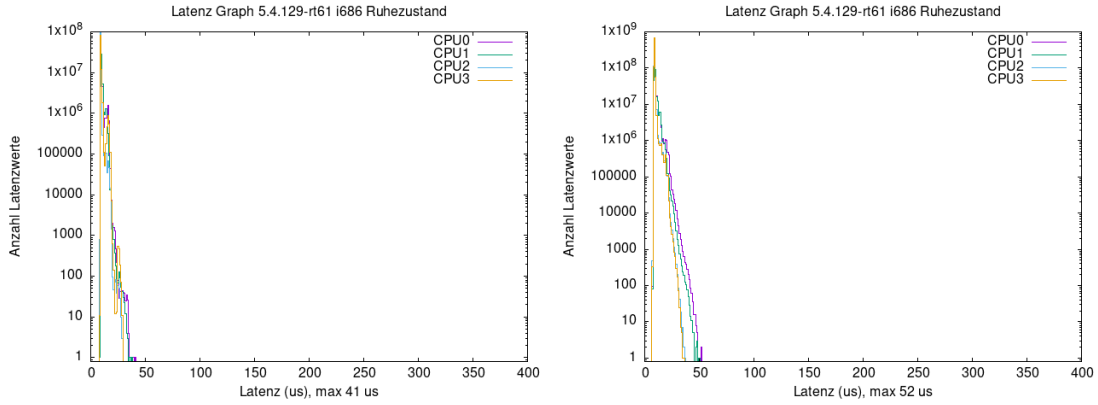
Im vorhergehenden Kapitel wurden die möglichen Alternativen für das Echtzeitbetriebssystem ergründet. Herauskristallisiert haben sich als Option der Preempt_RT Patch und Xenomai 3. Diese sollen in diesem Kapitel nun auf ihre Echtzeiteigenschaften untersucht werden. Dazu werden eigene Messungen geeigneter Kenngrößen zum einen mit Cyclictest und zum anderen mit dem Oszilloskop durchgeführt. Das Vorgehen ist hier analog zu dem Vorgehen bei Xenomai 2 in Abschnitt 3.3. Die exakten Konfigurationen der verwendeten Linux Kernel sind auf der beiliegenden CD zu finden.

5.1 Echtzeit Linux – Preempt_RT

Zuerst sollen die Echtzeitbetriebssysteme des Preempt_RT Patches ermittelt werden. Verwendet wird dazu der Linux Kernel in Version 5.4.129 mit RT-Patch Version 61. Betrachtet wird sowohl die 32Bit als auch die 64Bit Variante.

5.1.1 Performanzmessung mit Cyclictest

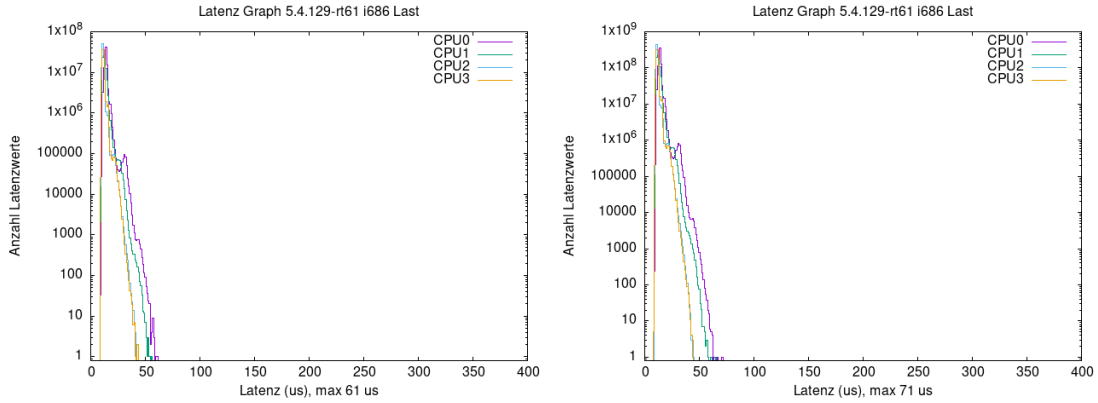
Zunächst wird die Schedulinglatenz mit Cyclictest gemessen. Die Ergebnisse für die Messung der 32Bit Variante im Leerlauf sind in Abbildung 5.1 dargestellt. Nach ca. sechs Stunden betrug die maximal gemessene Latenz 41µs und nach 48 Stunden 52µs.



(a) Idle 6h (b) Idle 48h

Abbildung 5.1: Cyclictest Messungen Preempt_RT 32bit Ruhezustand

Für die zweite Messung wird das System in den selben Lastmodus versetzt, wie er in Unterabschnitt 3.3.4 beschrieben wurde. Es wird ebenfalls über einen Zeitraum von sechs und 48 Stunden gemessen. Die Kurven sind in Abbildung 5.2 zu sehen.

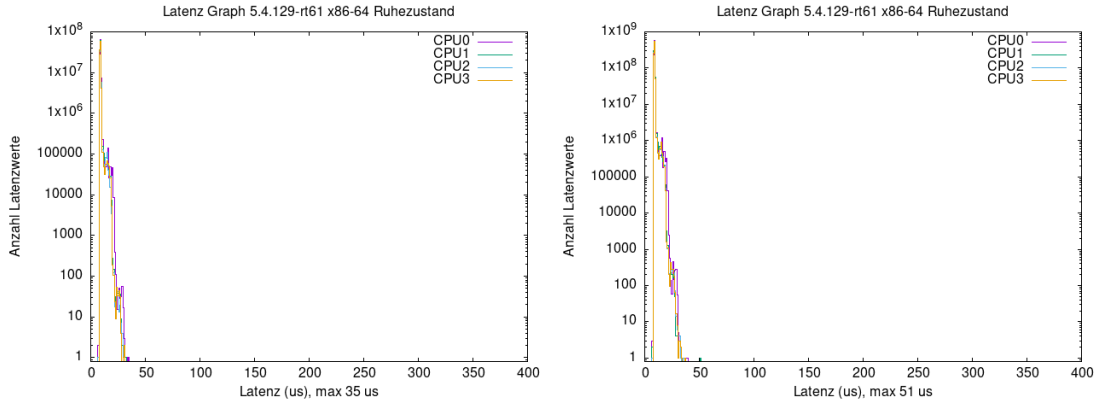


(a) Last 6h (b) Last 48h

Abbildung 5.2: Cyclictest Messungen Preempt_RT 32bit Last

Es ergeben sich für die Sechs-Stunden-Messung eine maximale Latenz von 61µs und für die 48-Stunden-Messung eine maximale Latenz von 71µs. Auch hier ist die Latenz für die längere Messung deutlich höher.

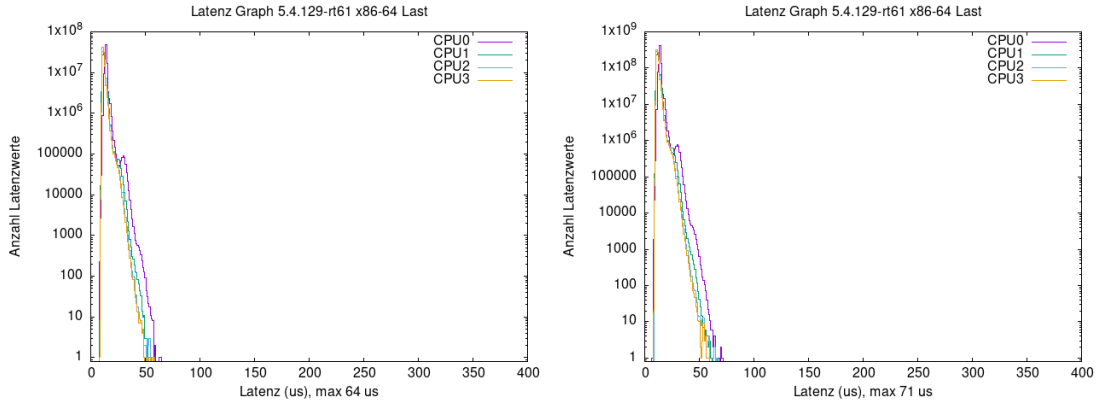
Für die Messung der 64Bit Variante im Leerlauf ergibt sich ein ähnliches Bild wie für die 32Bit Variante. Wie in Abbildung 5.3 zu sehen, betrug die maximale Latenz in der Sechs-Stunden-Messung 35µs und 51µs in der 48-Stunden-Messung.



(a) Idle 6h (b) Idle 48h

Abbildung 5.3: Cyclictest Messungen Preempt_RT 64bit Ruhezustand

Auch die Messung unter Last unterscheidet sich bei der 64Bit Variante in den Ergebnissen nur marginal von denen der 32Bit Variante wie Abbildung 5.4 verdeutlicht.



(a) Last 6h (b) Last 48h

Abbildung 5.4: Cyclictest Messungen Preempt_RT 64bit Last

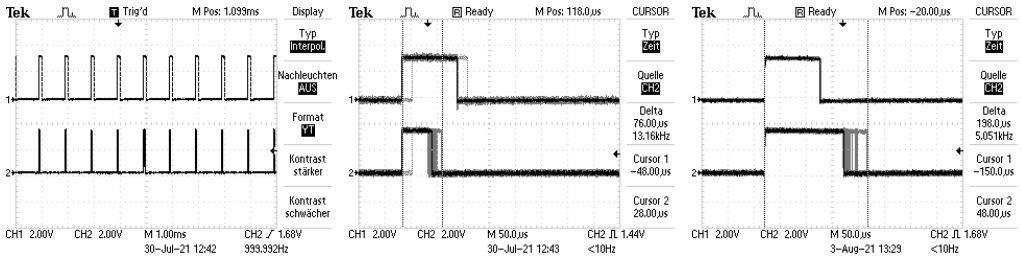
Ermittelt wurde nach sechs Stunden eine maximale Latenz von 64µs und 71µs nach 48 Stunden. Dass die Latenz bei den 48-Stunden-Messungen höher ausfällt, ist ein Indiz dafür, dass der von OSADL ausgeführte Sechs-Stunden-Test nicht ausreicht, um die

maximale Latenz zu ermitteln. Sie wird hier zu Zwecken der Vergleichbarkeit dennoch weiterhin ausgeführt.

5.1.2 Performanzmessung mit Oszilloskop

Die Performanzmessung mit dem Oszilloskop wird ebenso wie in Unterabschnitt 3.3.4 beschrieben durchgeführt.

Zuerst wird wieder die 32Bit Variante untersucht. Dazu wird zu Beginn die Funktionalität des Interrupts und des Pulsweiten-Triggers überprüft. Abbildung 5.5a und Abbildung 5.5b bestätigen die korrekte Konfiguration des FPGA sowie die korrekte Funktionalität des Oszilloskops.

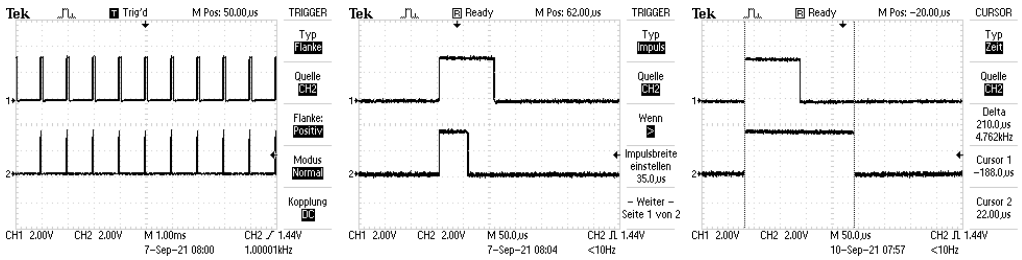


(a) Interrupt Test (b) Pulsweiten Test (c) Last 72h

Abbildung 5.5: Oszilloskop Messungen Preempt_RT 32Bit Last

Als Messzeitraum angedacht waren 96 Stunden. Wie Abbildung 5.5c zeigt, wurde bereits in einem Zeitraum von 72 Stunden eine maximale Interruptlatenz unter Last von 198µs gemessen und der Test abgebrochen. Rund 200µs entsprechen 20% des Zyklus der CNC Steuerungssoftware und rund dem Fünffachen der Latenz des Xenomai 2. Addiert man die Latenz auf die in Unterabschnitt 3.2.3 ermittelten maximalen Laufzeiten der Software, so besteht die Gefahr, dass häufiger Deadlines verpasst werden.

Wie bereits die Ergebnisse der Messungen mit Cyclictest angedeutet haben, ergibt sich für die 64Bit Variante ein ähnliches Bild wie für die 32Bit Variante. Um Fehler auszuschließen, wurde ebenfalls ein Test des Interrupts, zu sehen in Abbildung 5.6a und des Pulsweiten-Triggers, dargestellt in Abbildung 5.6a, durchgeführt. Unter Last ist auch die 64Bit Variante des Preempt_RT Patches nicht in der Lage die Erwartungen zu erfüllen und die Messung wird mit einer maximalen Latenz von 210µs erneut nach 72 Stunden abgebrochen.



(a) Interrupt Test (b) Pulseweiten Test (c) Last 72h

Abbildung 5.6: Oszilloskop Messungen Preempt_RT 64Bit Last

Als Zwischenbilanz kann also bereits festgehalten werden, dass die Interruptlatenz unter Linux mit Preempt_RT Patch in den angegebenen Versionen nicht den Ansprüchen der Steuerungssoftware genügt. Vergleicht man die Ergebnisse der Messungen mit Cyclictest mit denen von OSADL in Tabelle 2.1 und auf der Homepage[1], so gibt es auch keinen Grund zur Vermutung, dass eine andere Kombination an Linux Kernel und Preempt_RT Patch eine drastisch bessere Performanz bietet.

5.2 Xenomai 3

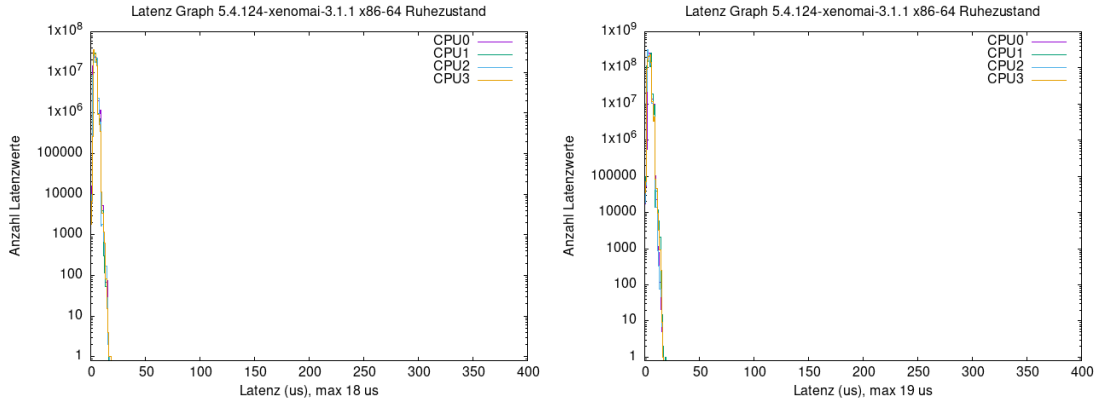
Nun soll das Echtzeitentwicklungsframework Xenomai in der Version 3 untersucht werden. Der ipipe-Patch steht nur für bestimmte Linux Kernel Versionen zur Verfügung. Für 32Bit wurde der Support nach der Kernel Version 4.9.146 eingestellt. Als Testversionen für die 32Bit Variante wurde somit Xenomai 3.1.1, der Linux Kernel in Version 4.9.146 und ipipe-core-4.9.146-x86-8 verwendet. In dieser Konstellation führte das in Unterabschnitt 3.3.4 beschriebene Lastszenario reproduzierbar zu einer Kernel Panik. Auf ältere Versionen zurückzugreifen widerspricht der Aufgabenstellung der Aktualisierung des in die Jahre gekommenen Betriebssystems der CNC Steuerung. Es muss also festgestellt werden, dass Xenomai 3 unter 32Bit wegen der eingestellten Entwicklung keine Option darstellt.

Für die 64Bit Variante wird Xenomai in Version 3.1.1 und der Linux Kernel in Version 5.4.124 verwendet. Der ipipe-Patch kommt in Version ipipe-core-5.4.124-x86-5 zum Einsatz.

[1] https://www.osadl.org/Latency-plots.latency-plots.0.html

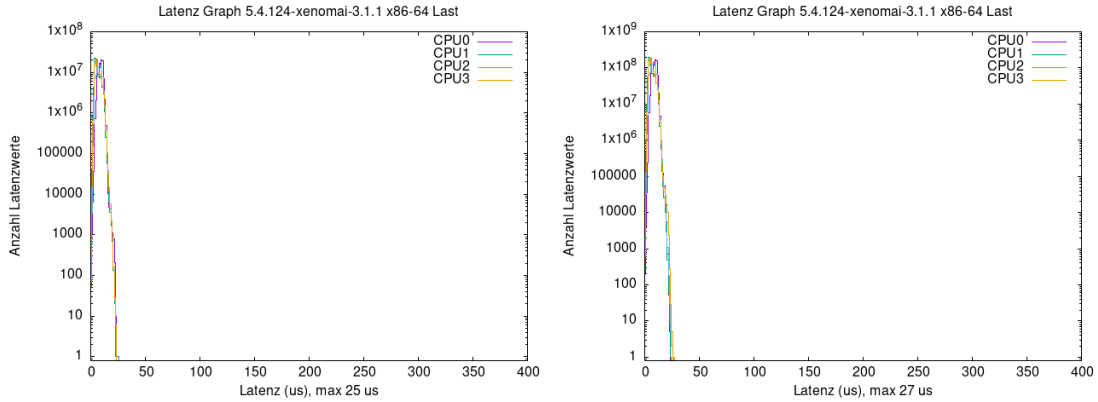
5.2.1 Performanzmessung mit Cyclictest

Zuerst wird wie gehabt die Schedulinglatenz mit Cyclictest ermittelt. Die Sechs-Stunden-Messung unter Leerlaufbedingungen, zu sehen in Abbildung 5.7a, ergibt eine maximale Latenz von 18µs. Selbst im Zeitraum von 48 Stunden erhöht sich diese Latenz nur marginal auf 19µs, dargestellt in Abbildung 5.7b.



(a) Idle 6h (b) Idle 48h

Abbildung 5.7: Cyclictest Messungen Xenomai3 Ruhezustand



(a) Last 6h (b) Last 48h

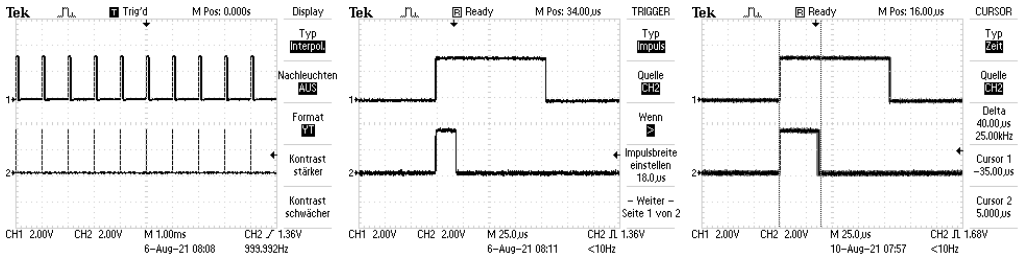
Abbildung 5.8: Cyclictest Messungen Xenomai3 Last

Auch unter dem gewählten Last-Szenario erhöht sich die Latenz bei der Sechs-Stunden-Messung nur leicht auf 25µs und nach 48 Stunden auf 27µs. Der Graph für die Sechs-Stunden-Messung ist in Abbildung 5.8a zu sehen und für die 48-Stunden-Messung in

Abbildung 5.8b. Diese Messergebnisse stellen, wie in Unterabschnitt 2.5.2 beschrieben, eine optimistische Einschätzung dar. Dennoch erfüllt eine Schedulinglatenz von rund 30µs voll und ganz die Anforderungen der CNC Steuerung.

5.2.2 Performanzmessung mit Oszilloskop

Zum Schluss folgt noch die Messung der Interruptlatenz mit dem Oszilloskop. Das Prozedere wird exakt wie zuvor schon bei Xenomai 2 und dem Preempt_RT Patch durchgeführt. Es wird wieder das selbe Lastszenario gewählt und eine Überprüfung der korrekten Konfiguration des FPGA durchgeführt. Die richtig eingestellten Interrupts sind in Abbildung 5.9a zu sehen. Die Funktion des Pulsweiten Triggers ist in Abbildung 5.9b zu betrachten.



(a) Interrupt Test (b) Pulseweiten Test (c) Last 96h

Abbildung 5.9: Oszilloskop Messungen Xenomai3 Last

In dem gesamten Testzeitraum von 96 Stunden betrug die maximale Interruptlatenz unter Last 40µs. Das Bild dazu ist in Abbildung 5.8 zu sehen. Dieses Ergebnis entspricht exakt den 40µs der zuvor getesteten und im Einsatz befindlichen Vorgängerversion Xenoami 2. Es ist also davon auszugehen, dass ein Umstieg auf Xenomai 3 die selbe Performanz bieten wird wie Xenomai 2.

6 Evaluation

In diesem Kapitel werden die Ergebnisse der Untersuchung aufbereitet und zusammengefasst. Basierend auf diesen Daten sowie den Anforderungen aus Kapitel 4 werden die Konsequenzen bei einem Umstieg auf das entsprechende Echtzeitbetriebssystem diskutiert und dann eine Empfehlung für ein Nachfolgersystem ausgesprochen.

6.1 Zusammenfassung der Messergebnisse

Die Ergebnisse der Messungen der Echtzeitbetriebssysteme mit Cyclictest und des Oszilloskops sind in Tabelle 6.1 dargestellt. Zunächst wird der Xenomai 2 des bestehenden Systems betrachtet und als Messlatte verwendet. Die ermittelten Werte liegen bei der Idle Messung mit Cyclictest im ersten Versuch bei 25µs und im zweiten Versuch bei 7µs. Hierbei wurde wegen der Unzuverlässigkeit des Cyclictest-Tools auf dieser Plattform, wie in Kapitel 3 beschrieben, auf weitere Messungen verzichtet. Die maximal auf dem Oszilloskop gemessene Interruptlatenz liegt bei 40µs. Dies entspricht 4% der Zyklusdauer von 1ms.

Nun werden die Messergebnisse für den Preempt_RT unter 64Bit und unter 32Bit verglichen. Die Messergebnisse liegen zwischen 35µs und 71µs und besitzen somit ein Delta von 36µs. Eine erste Erkenntnis ist, dass bei der 48h-Lastmessung unter 32Bit und unter 64Bit die Latenzen mit 71µs identisch sind. Ebenso sind kaum Unterschiede der restlichen Messungen festzustellen. Das Delta bei der 48h Idlemessung beträgt lediglich 1µs und 3µs bei der 6h Lastmessung. Unter Last beträgt das Delta der Latenzen bei der 6h und der 48h Messungen 7µs bei 64Bit bzw. 10µs bei 32Bit.

Die Ergebnisse bei der Interruptlatenz-Messung fallen bei beiden mit 210µs bzw 198µs recht hoch aus. Dies entpricht bei beiden rund 20% der Zyklusdauer von 1ms. Es kann also geschlussfolgert werden, dass es beim Preempt_RT in Bezug auf die Echtzeitperformanz keine Rolle spielt ob eine 64Bit oder eine 32Bit Variante gewählt wird.

	Xenomai 2 32Bit	Xenomai 3 64Bit	Preempt_RT 32Bit	Preempt_RT 64Bit
Cyclictest Leerlauf 6h in µs	7 bzw 25	18	41	35
Cyclictest Leerlauf 48h in µs		25	52	51
Cyclictest Last 6h in µs		19	61	64
Cyclictest Last 48h in µs		27	71	71
Oszilloskop Last in µs	40	40	198	210

Tabelle 6.1: Übersicht über die Performanz-Messungen

Wie in Kapitel 4 beschrieben, besteht bei Xenomai 3 keine Option auf eine 32Bit Variante. Daher wurde die Untersuchung auf 64Bit beschränkt. Die Messergebnisse mit Cyclictest liegen zwischen 18µs und 27µs und haben somit ein Delta von 9µs. Betrachtet man das sehr geringe Delta von 2µs zwischen der 6h-Lastmessung und der 48-Stunden-Lastmessung, kann davon ausgegangen werden, dass die Performanz sehr konstant ist. Das Ergebnis der Oszilloskop Messung ist mit 40µs identisch mit dem Ergebnis der Vorgängerversion Xenomai 2. Auch hier lassen sich also die Xenomai Versionen in Bezug auf die Echtzeitperformanz zusammenfassen.

6.2 Diskussion der Ergebnisse

Nachdem nun festgestellt werden konnte, dass sich Xenomai 2 mit Xenomai 3 ebenso wie Preempt_RT unter 32Bit und 64Bit in punkto Echtzeitperformanz zusammenfassen lassen, können nun der Preempt_RT und Xenomai zueinander in Relation gesetzt werden. 35-71µs beim Preempt_RT gegenüber 18-27µs beim Xenomai mit Cyclictest bedeutet eine zwei- bis dreimal höhere Latenz bei ersterem. Bei der Oszilloskop Messung stehen sich 40µs und rund 200µs gegenüber. Dies entspricht einer fünffach höheren maximalen Latenz bei Preempt_RT. Bezieht man nun das zeitliche Verhalten der CNC Steuerung mit ein, ergibt sich ein recht eindeutiges Bild. Die maximal gemessene Laufzeit im Anwendungsfall Peck betrug 721µs und die ermittelte potentielle maximale Laufzeit lag bei 930µs. Wird nun die ermittelte Interruptlatenz diesen Laufzeiten hinzuaddiert, so ergeben sich 761µs bzw. 970µs für Xenomai 3 und 921µs bzw 1,130µs für den Preemt_-RT Patch. Berücksichtigt muss ebenfalls werden, dass von dem 1ms Takt 100µs für die DMA-Übertragung reserviert sind.

Es ist klar zu sehen, dass Linux mit Preempt_RT Patch sowohl für die maximal gemessene Laufzeit als auch für das theoretisch berechnete Maximum das Timing mit einer realistischen Chance verfehlt. Ein Verlust von rund 20% Zeit innerhalb des Zyklus des Systems ist nicht hinnehmbar und schließt Linux mit Preempt_RT als möglichen Kandidaten aus. Die Ergebnisse sind an dieser Stelle so eindeutig, dass ein Einbeziehen der Erkenntnisse aus der Caching- und Branching-Analyse nicht nötig ist.

Für die Einschätzung des Xenomai 3 ergibt sich ein anderes Bild. Hier bleibt die maximal gemessene Laufzeit inklusive der Interruptlatenz mit 761µs mit noch relativ viel Abstand zu der erlaubten Obergrenze von 900µs. Dies gilt für alle ermittelten Laufzeiten der untersuchten CNC Anwendungsfälle. Die ermittelten maximalen Laufzeiten stellen jedoch nicht die tatsächliche WCET dar. Eine tatsächliche WCET konnte aus den in Kapitel 2 aufgeführten Gründen nicht ermittelt werden. Daher wurde aus den maximalen Laufzeiten der Einzelabschnitte eine potentielle Obergrenze von 930µs errechnet. Diese Obergrenze lässt mit den maximal 40µs Interruptlatenz des Xenomai 3 nur noch theoretische 30µs Zeit für die DMA-Übertragung, was nicht ausreichend ist. Selbst wenn die Interruptlatenz bei 0 liegen würde, wäre die Marke von 900µs bereits überschritten. In sehr seltenen Fällen besteht also die Gefahr, dass das Timing nicht eingehalten wird und die Servomotoren nicht rechtzeitig mit neuen Daten versorgt werden können. Dieser Umstand wurde Sieb & Meyer mitgeteilt und fließt bereits in die Entwicklung der CNC Steuerungssoftware mit ein. Xenomai 3 ist in Punkto Echtzeitperformanz identisch mit der Vorgängerversion und bildet damit die exakt gleiche Grundlage für die CNC Steuerungssoftware wie das bisher verwendete Echtzeitbetriebssystem. Somit ist Xenomai 3 die logische Wahl für das zukünftige Betriebssystem.

Dennoch haben die Untersuchungen der CNC Steuerungssoftware neue Erkenntnisse gebracht, die berücksichtigt werden sollten. Die Ergebnisse der Caching- und Branching-Analyse haben ergeben, dass die CNC Steuerungssoftware über 97% Hits im LLC besitzt und Verzweigungen in nur unter 9% der Fälle falsch vorhergesagt werden. Es ist also davon auszugehen, dass die ermittelten Laufzeiten leicht optimistisch sind. Bei der Weiterentwicklung muss daher darauf geachtet werden, wie sich Änderungen der Software auf das Caching- und Branching-Verhalten auswirken. Hier soll noch einmal der Umstand aus Abbildung 2.6 ins Gedächtnis gerufen werden, dass der Sprung von 97% Hits zu 96% Hits einen deutlich größeren Einfluss auf die Performanz hat als bei einem Sprung von 11% auf 10%.

7 Zusammenfassung und Ausblick

Nun sollen noch einmal die Ergebnisse zusammengefasst werden. Den Abschluss bildet schließlich ein Ausblick in die Zukunft.

7.1 Zusammenfassung

Ziel dieser Arbeit war es, einen passenden, aktuellen Ersatz für das Echtzeitbetriebssystem einer CNC Steuerung zu ermitteln und den Portierungsaufwand abzuschätzen.

Dazu wurden die folgenden Schritte durchgeführt:

Ermitteln des Aufbaus der CNC Steuerung Über einen PC, den Communication Controller, wird Bohr-/Fräsprogramm über Ethernet an den Motion Controller gesendet. Dieser puffert die Daten und wandelt den Code in Instruktionen für die Servomotoren der CNC Maschine um. Die Instruktionen werden in Puffern so zusammengefasst, dass nach jedem Bohrloch ein sicherer Zustand erreicht wird und sie als atomare Einheit abgearbeitet werden können. Teil des Motion Controllers ist ein FPGA, welches über den PCIe Bus eingebunden ist und als Taktgeber für das System dient. Dieses FPGA ist ebenso über einen digitalen Bus per Lichtwellenleiter mit den Servoverstärkern und dem Frequenzumrichter verbunden. Die Motion Controller Software erhält alle 1ms einen Interrupt von dem FPGA und schreibt in diesem Takt die berechneten Instruktionen in den DMA-Speicher des FPGAs. Danach überträgt das FPGA die Daten über den digitalen Bus an die Steuerung der Servomotoren. Sind die Instruktionen abgearbeitet, wird dies zurück an den Communication Controller gemeldet und der nächste Takt beginnt.

Messung der Scheduling- und Interruptlatenz des alten Betriebssystems Das Betriebssystem des Motion Controller basiert auf dem Linux Kernel in Version 3.14.17 mit 32Bit und verwendet den Cokernel des Echtzeitentwicklungsframeworks Xenomai in Version 2.6.4. Zur Ermittlung der Scheduling Latenz wurde das Tool Cyclictest verwendet. Herausgestellt hat sich jedoch, dass die im Xenomai Framework enthaltene Version unzuverlässig arbeitet und zu Abstürzen führt. In zwei Versuchen wurden Latenzen von 7µs bzw. 25µs unter Leerlaufbedingungen gemessen.

Für die Ermittlung der Interruptlatenz wurde das FPGA so programmiert, dass alle 1ms ein Interrupt ausgelöst und ein Pin auf High gesetzt wird. Die Motion Controller Software reagiert auf diesen Interrupt und setzt den Pin wieder auf Low. Mit einem Oszilloskop und einem Pulsweiten-Trigger wurde gemessen, wie lange es dauert, bis der Pin wieder auf Low steht. Unter einem definierten Lastszenario, welches einer realistischen maximalen Auslastung des Systems entspricht, wurde eine Interruptlatenz von 40µs gemessen.

Daraus Ableiten der Anforderungen an ein neues Betriebssystem Durch den Aufbau des Systems und Vorgaben von Sieb & Meyer konnten die nachfolgenden Anforderungen an ein neues Echtzeitbetriebssystem ermittelt werden. Folgende Punkte stellen die primären Anforderungen dar:

- Unterstützung für Intel x86 Architektur
- Ethernet und TCP/IP
- CIFS/NFS Unterstützung
- PCIExpress/DMA
- Multithreading Support
- Frei verfügbar/nicht proprietär
- sehr gute Debug Möglichkeiten
- Unterstützung für SSH

Als sekundäre Anforderungen können weitere Punkte identifiziert werden:

- 32Bit Support
- Zukunftssicher

- Unterstützung für weitere Plattformen

Suche nach neuen Kandidaten für das Echtzeitbetriebssystem inklusive Aufwandsabschätzung Es gibt eine ganze Reihe an Echtzeitbetriebssystemen. Wikipedia[1] und die Internetseite osrtos[2] liefern hier eine große Auswahl. Reduziert man die Optionen jedoch um alle Proprietären Systeme und alle Systeme, welche keine x86 Plattformen unterstützen, so verkürzt sich die Liste beträchtlich. Entfernt man nun weiter alle Betriebssysteme, welche seit mehr als einem Jahr keine Updates erhalten haben und jene, welche zu spezialisierte Anwendungsbereiche im Fokus haben, so bleiben nur zwei primäre Optionen übrig.

Xenomai in Version 3 ist die logische Updatemöglichkeit und erfüllt alle Anforderungen. Es muss lediglich von Version 2 nach 3 portiert werden. Xenomai hat jedoch den Support für 32Bit unter Intel Plattformen eingestellt, daher müssen alle Komponenten der CNC Steuerungssoftware zusätzlich nach 64Bit portiert werden. Geringer bis mittlerer Portierungsaufwand.

Linux mit Preempt_RT Patch erfüllt ebenso alle Anforderungen und hat gegenüber Xenomai den Vorteil, dass weniger Anpassungen an den Linux Kernel nötig sind. Ebenso wandern mit jeder neuen Kernelversion mehr Teile des Preempt_RT Patches in die Mainline des Linuxkernels. Es müssen gegenüber Xenomai keine speziellen APIs verwendet werden. Da die CNC Steuerungssoftware und der Treiber für das FPGA die API von Xenomai 2 verwenden, müssen diese Aufrufe durch die Linux API ersetzt werden. Geringer bis mittlerer Portierungsaufwand.

Sie erfüllen alle Anforderungen und verlangen den kleinsten Portierungsaufwand. Durch die Verwendung eines linuxbasierten Ansatzes ist die Unterstützung für die Hardware ideal, die Dokumentation hervorragend und die zur Verfügung stehende Software vielfältig.

Die Echtzeitbetriebssysteme NuttX, Zephyr und seL4 verbleiben als sekundäre Kandidaten und werden untersucht, sollten die primären Optionen in Punkto Echtzeiteigenschaften die Anforderungen nicht erfüllen.

[1] https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems
[2] https://www.osrtos.com/

Performanzmessung der Kandidaten Bei der Performanzmessung von Xenomai 3 und des Preempt_RT ist das Vorgehen exakt dasselbe wie zuvor bei Xenomai 2. Das Lastszenario wird identisch gewählt und der Versuchsaufbau für die Interruptlatenzmessung entspricht ebenfalls dem Aufbau wie zuvor bei Xenomai 2. Gemessen wurden jeweils die 64Bit Varianten der Betriebssysteme und die 32Bit Variante für den Preempt_RT. Verwendet wurde der Kernel 5.4.129, rt61 für den Preempt_RT. Für Xenomai 3 wurde der Kernel 5.4.124, Xenomai 3.1.1 und ipipe-core-5.4.124 verwendet. Die maximale Scheduling Latenz unter Last betrug für den Preempt_RT 71µs für 32Bit und 64Bit und 27µs für Xenomai 3. Bei der Messung der Interruptlatenz des Preempt_RT traten bereits nach wenigen Stunden Latenzen von ca. 200µs auf, so dass die angedachte Testzeit von 96 Stunden vorzeitig abgebrochen wurde. Die Interruptlatenz des Xenomai 3 betrug in diesem Zeitraum von 96 Stunden maximal 40µs.

Auswahl des am besten geeigneten Systems Waren die Testergebnisse mit Cyclictest noch vielversprechend für beide Echtzeitbetriebssysteme, so ergibt die Messung der Interruptlatenz ein überraschend negatives Ergebnis für den Preempt_RT. Eine Interruptlatenz von 200µs entspricht rund 20% Zeitverlust im 1ms Zyklus des CNC Systems. Noch mehr, wenn man die 100µs vom Zyklus abzieht, die für den DMA-Transfer reserviert sind. Addiert man diese 200µs Latenz auf die maximal gemessene Ausführungszeit von 721µs, so ist die Deadline von 900µs überschritten. Wird nun auch noch das Ergebnis der Branching- und Caching-Analyse mit einbezogen, so muss davon ausgegangen werden, dass sich die Zeiten eher verschlechtern können als verbessern. Es muss also der Schluss gezogen werden, dass die CNC Steuerungssoftware in der verwendeten Version und Konfiguration unter Linux mit Preempt_RT Patch die zeitlichen Anforderungen nicht erfüllt.

Die Interruptlatenz des Xenomai 3 entspricht exakt der Interruptlatenz des aktuell verwendeten Xenomai 2 und erfüllt damit voll und ganz die Anforderung. Dennoch hat die Auswertung der maximal gemessenen Latenzen ergeben, dass Ausreißer mit einer hohen Auslastung des 1ms Zyklus vorkommen. Ebenfalls legt, wie oben erwähnt, das Ergebnis der Branching- und Caching-Analyse nahe, dass die Caches von der Software aktuell gut ausgenutzt werden und sich daher die Ausführungszeit kaum verbessern, jedoch stark verschlechtern kann. Als weiterer Aspekt ist bei der Verwendung von Xenomai 3 noch zu beachten, dass dieser nur noch in der 64Bit Version verwendet werden kann. Hier gilt es zusätzlichen Portierungsaufwand zu betreiben.

7.2 Ausblick

Auch wenn mit Xenomai 3 ein geeigneter Ersatz für das Betriebssystem der CNC Steuerung gefunden werden konnte, so bleiben dennoch einige Punkte offen. Durch den erzwungenen Umstieg auf 64Bit muss untersucht werden, wie sich dies auf den Speicherbedarf der Steuerung auswirkt und welchen Einfluss dies auf die Ausführungszeiten hat. Xenomai 3 unterstützt bedingt durch den ipipe Patch maximal Linux Kernel der Version 5.4. Zum Zeitpunkt des Schreibens dieser Arbeit liegen bereits zwei neue Hardwarerevisionen der CNC Steuerung bereit und es gilt auszuloten, wie gut die Hardwareunterstützung dieser Kernelversion für diese Boards ist.

DENX Software Engineering, die Firma hinter Xenomai, hat bereits die Arbeit an Xenomai 4 aufgenommen. Hier findet ein Architekturwechsel weg von der Interrupt-Pipeline ipipe hin zu Dovetail statt. Ebenso wird, beginnend mit Xenomai 3.3, an einer auch für zukünftige Versionen gemeinsamen Sammlung an Features und Interfaces gearbeitet. Es bleibt abzuwarten, welche Auswirkungen dies bei einem weiteren Upgrade der Xenomai Version mit sich bringt.

Die Analyse des zeitlichen Verhaltens der CNC Steuerungssoftware hat ergeben, dass die durchschnittliche Last sehr gering ist, es jedoch in regelmäßigen Abschnitten hohe Ausschläge gibt. Es wird bereits daran gearbeitet, die Berechnungen, welche diese Spitzen verursachen, je nach Möglichkeit besser aufzuteilen. Auch die Branching- und Caching-Analyse mit perf hat neue Erkenntnisse zum Verhalten der Software gebracht. Erneute Messungen nach Änderungen im Code würde einen Vergleich zulassen und deren Ergebnisse Indikatoren für eine evtl. Verschlechterung der Laufzeiten darstellen. Durch eine Einbeziehung des Quellcodes ließe sich mit perf ebenfalls eine genauere Analyse durchführen und damit evtl. das Vorhersagen von Branches verbessern. Eine Einbindung und evtl. Automatisierung dieser beiden Analyseverfahren in den Entwicklungsprozess könnte eine weitere Maßnahme sein.

Zuletzt muss die CNC Steuerungssoftware noch auf das neue System portiert werden und der endgültige Beweis, dass Xenomai 3 das etablierte Betriebssystem ersetzen kann, erbracht werden.

Literaturverzeichnis

[1] Abel, Andreas ; Reineke, Jan: Reverse engineering of cache replacement policies in Intel microprocessors and their evaluation. In: *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, S. 141–142

[2] Bernat, G. ; Colin, A. ; Petters, S.M.: WCET analysis of probabilistic hard real-time systems. In: *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, 2002, S. 279–288

[3] Bovet, Daniel P. ; Cesati, Marco: *Understanding the Linux Kernel*. 3rd Edition. O'Reilly Media, Inc., 2005. – ISBN 9780596005658

[4] Brown, Jeremy ; Martin, Brad: How fast is fast enough? Choosing between Xenomai and Linux for real-time applications. In: *Proceedings of the 12th Real-Time Linux Workshop (RTLWS'12)* (2010), 04

[5] Chang, Chen-Chung ; Hsia, Shao-Yi ; Huang, Hung-Di: Improvement on CNC drilling quality using vibration suppression fixture. In: *2018 IEEE International Conference on Applied System Invention (ICASI)*, 2018, S. 910–913

[6] Coleman, James: Reducing Interrupt Latency Through the Use of Message Signaled Interrupts / Intel corporation. Januar 2009 (321070). – Forschungsbericht.

[7] Corti, Matteo ; Brega, Roberto ; Gross, Thomas: Approximation of Worst-Case Execution Time for Preemptive Multitasking Systems. In: Davidson, Jack (Hrsg.) ; Min, Sang L. (Hrsg.): *Languages, Compilers, and Tools for Embedded Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2001, S. 178–198. – ISBN 978-3-540-45245-4

[8] Emde, Carsten: Long-term monitoring of apparent latency in PREEMPT RT Linux realtime systems. In: *RTLWS12* (2010)

[9] Garre, Carlos ; Mundo, Domenico ; Gubitosa, Marco ; Toso, Alessandro: Real-time and real-fast performance of general-purpose and real-time operating systems in multithreaded physical simulation of complex mechanical systems. In: *Mathematical Problems in Engineering* 2014 (2014)

[10] Gregg, Brandon: *Systems Performance - Enterprise and the Cloud*. Pearson Education Canada, 2020 (Addison-Wesley Professional Computing Series). – ISBN 9780136820154

[11] Hennessy, J.L. ; Patterson, D.A.: *Computer Architecture: A Quantitative Approach*. Elsevier Science, 2017 (ISSN). – ISBN 9780128119068

[12] Intel Corporation: *Intel Atom Processor E3800 Product Family - Datasheet*. 1. Santa Clara, Vereinigte Staaten: Intel Corporaten (Veranst.), 2013

[13] Intel Corporation: *Intel 64 and IA-32 Architectures Software Developerś Manual*. Volume 3B. Santa Clara, California: Intel Corporation (Veranst.), 2016

[14] Jonathan Corbet, Greg Kroah-Hartman: *Linux Device Drivers*. 3rd Edition. O'Reilly Media, Inc., 2005. – ISBN 9780596005900

[15] Labrosse, Jean: *MicroC/OS-II: The Real Time Kernel*. CRC Press, 2002. – ISBN 1-57820-103-9

[16] Lee, Edward A.: Cyber Physical Systems: Design Challenges. In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, S. 363–369

[17] Levinthal, David: Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors / Intel corporation. 2009. – White Paper.

[18] Lv, Mingsong ; Guan, Nan ; Reineke, Jan ; Wilhelm, R. ; Yi, W.: A Survey on Static Cache Analysis for Real-Time Systems. In: *Leibniz Transactions on Embedded Systems* 3 (2016), S. 05:1–05:48

[19] Lyu, Xinchen ; Ren, Chenshan ; Ni, Wei ; Tian, Hui ; Liu, Ren P.: Cooperative Computing Anytime, Anywhere: Ubiquitous Fog Services. In: *IEEE Wireless Communications* 27 (2020), Nr. 1, S. 162–169

[20] Macarenco, Konstantin ; Frye, Kristina ; Hamlin, Benjamin ; Karavanic, Karen L.: The Effects of System Management Interrupts on Multithreaded, Hyperthreaded, and MPI Applications. In: *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, 2016, S. 338–345

[21] Murikipudi, Akhilesh ; Prakash, V ; Vigneswaran, T: Performance analysis of real time operating system with general purpose operating system for mobile robotic system. In: *Indian Journal of Science and Technology* 8 (2015), Nr. 19, S. 1–6

[22] PCI-SIG: *PCI Express Base Specification.* Revision 2.1. Beaverton, Oregon: PCI-SIG (Veranst.), 2009

[23] Reghenzani, Federico ; Massari, Giuseppe ; Fornaciari, William: Mixed Time-Criticality Process Interferences Characterization on a Multicore Linux System. In: *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, S. 427–434

[24] Reghenzani, Federico ; Massari, Giuseppe ; Fornaciari, William: The Real-Time Linux Kernel: A Survey on PREEMPT_RT. In: *ACM Comput. Surv.* 52 (2019), Februar, Nr. 1. – URL https://doi.org/10.1145/3297714. – ISSN 0360-0300

[25] Schmidt, Albrecht: Ubiquitous Computing: Are We There Yet? In: *Computer* 43 (2010), Nr. 2, S. 95–97

[26] Seneta, Eugene B. ; Young, John S. ; Buscher, David F. ; Ligon, E. R. ; Etscorn, Dylan ; Farris, Allen: When you want it done right now: experience from programming hard real time systems in Xenomai for the Magdalena Ridge Observatory interferometer. In: Guzman, Juan C. (Hrsg.) ; Ibsen, Jorge (Hrsg.): *Software and Cyberinfrastructure for Astronomy VI* Bd. 11452 International Society for Optics and Photonics (Veranst.), SPIE, 2020, S. 245 – 256. – URL https://doi.org/10.1117/12.2562191

[27] Serino, Anthony ; Cheng, L.: A Survey of Real-Time Operating Systems. 2019. – Forschungsbericht.

[28] Sieb & Meyer AG: *95-00 Technisches Handbuch.* 1. Lüneburg, Germany: Sieb & Meyer AG (Veranst.), 2021

[29] Sivanich, Dimitri: Low latency real-time computing on multiprocessor systems running standard Linux. In: *Proceedings of 11th High Performance Embedded Computing (HPEC'07) Workshop*, 2007

[30] Stankovic, John ; Ramamritham, Krithivasan: What is Predictability for Real-Time Systems? In: *Real-Time Systems* 2 (1990), 11, S. 247 – 254

[31] Steinbach, Till ; Korf, Franz ; Schmidt, Thomas C.: Simulation und Evaluation von Echtzeit-Ethernet in Fahrzeugnetzen. In: *PIK - Praxis der Informationsverarbeitung und Kommunikation* 35 (2012), Mai, Nr. 2, S. 67–74. – ISSN 0930-5157

[32] Tektronix Inc.: *TDS1000 and TDS2000 Series Digital Storage Oscilloscopes - User Manual*. 1. Beaverton, Vereinigte Staaten: Tektronix Inc. (Veranst.), 2005

[33] Ugal, AS: Hard Real Time Linux using Xenomai on Intel Multi-Core Processors. In: *Intel corporation* (2009)

[34] Wilhelm, Reinhard ; Engblom, Jakob ; Ermedahl, Andreas ; Holsti, Niklas ; Thesing, Stephan ; Whalley, David ; Bernat, Guillem ; Ferdinand, Christian ; Heckmann, Reinhold ; Mitra, Tulika ; Mueller, Frank ; Puaut, Isabelle ; Puschner, Peter ; Staschulat, Jan ; Stenström, Per: The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools. In: *ACM Transactions on Embedded Computer Systems* 7 (2008), may, Nr. 3. – URL https://doi.org/10.1145/1347375.1347389. – ISSN 1539-9087

[35] Yaghmour, Karim: Adaptice Domain Environment for Operating Systems. 2001. – Forschungsbericht.

[36] Yaghmour, Karim ; Masters, Jon ; Ben-Yossef, Gilad ; Gerum, Philippe: *Building embedded Linux systems*. 2nd Edition. O'Reilly Media, Inc., 2008. – ISBN 9780596529680

[37] Yang, Chung-Fan ; Shinjo, Yasushi: Obtaining Hard Real-Time Performance and Rich Linux Features in a Compounded Real-Time Operating System by a Partitioning Hypervisor. In: *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. New York, NY, USA : Association for Computing Machinery, 2020 (VEE '20), S. 59–72. – URL https://doi.org/10.1145/3381052.3381323. – ISBN 9781450375542

A Anhang

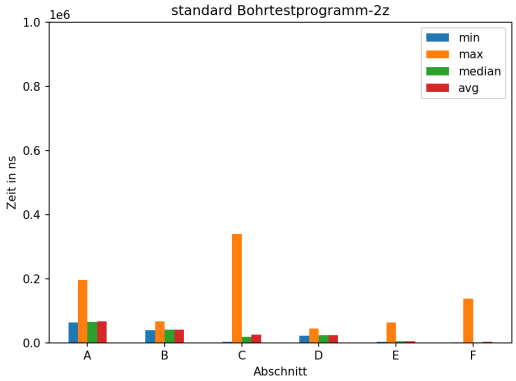


Abbildung A.1: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Standard Bohrtestprogramm 2z

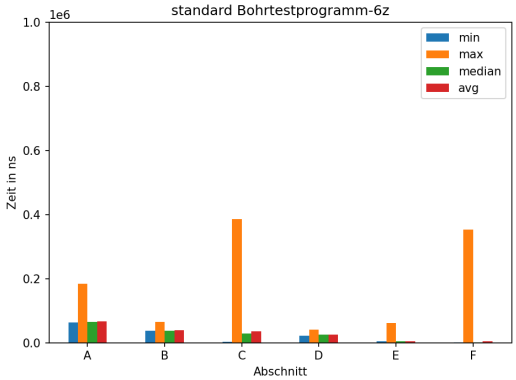


Abbildung A.2: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Standard Bohrtestprogramm 6z

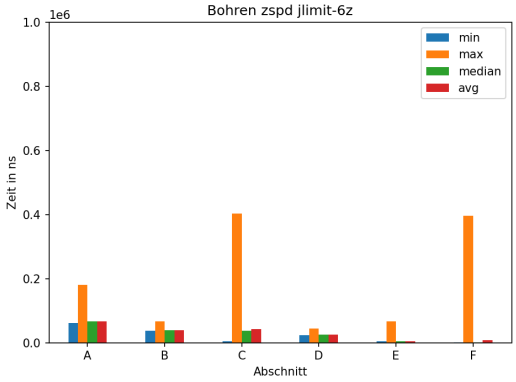


Abbildung A.3: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Bohren mit zsp und jlimit mit 6z

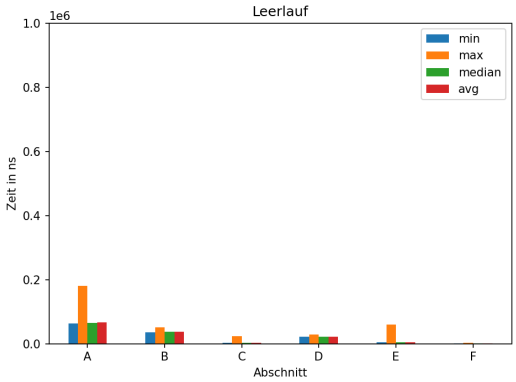


Abbildung A.4: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Leerlauf

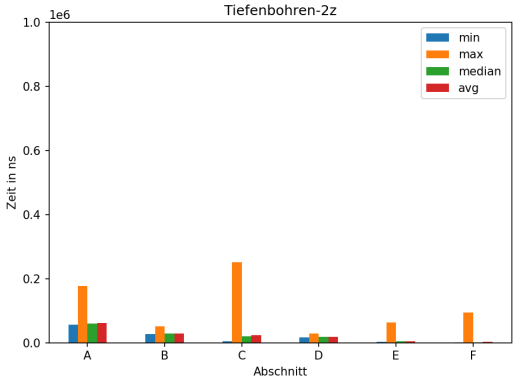


Abbildung A.5: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Tiefenbohren mit 2z

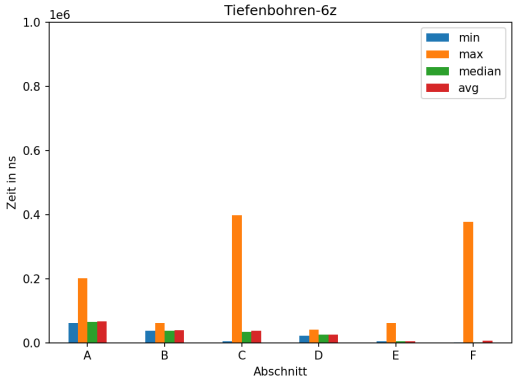


Abbildung A.6: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Tiefenbohren mit 6z

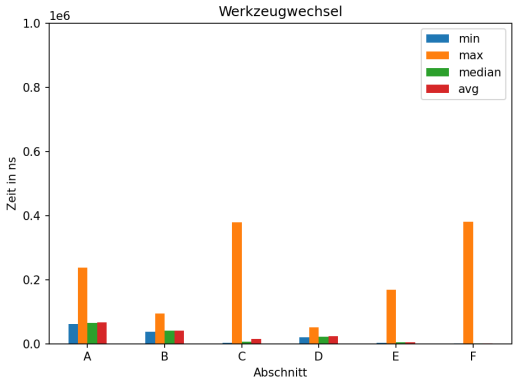


Abbildung A.7: Laufzeiten der Abschnitte A bis F für den Anwendungsfall Werkzeugwechsel

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original