Universität Leipzig
Fakultät für Physik und Geowissenschaften

Erweitern eines Gentoo-Linux mit RTAI, LabVIEW und Comedi als digitaler Regler

Bachelorarbeit

Leipzig, den 04.02.2008

vorgelegt von

Markus Jäger

geb. am: 04.02.1983

Studiengang: Bachelor Physik

Betreuung: Juniorprof. Dr. Petrik Galvosas

Für Anne

Danksagung

Mit diesen Worten möchte ich allen danken, die mir diese Arbeit ermöglicht haben.
Ich danke Juniorprof. Dr. Petrik Galvosas für das interessante Arbeitsthema und für seine großzügige Hilfbereitschaft und Betreuung während der Ausarbeiung.
Ebenfalls gilt großer Dank meiner Freundin Anne, die mit ihrer zeitlichen Unterstützung diese Arbeit möglich gemacht hat.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der Einsatz von Computern, zur unterstützenden Durchführung physikalischer Experimente, gewinnt immer mehr an Bedeutung. Computer, unter anderem auch **P**ersonal **C**omputer (kurz PC), können durch ihre heutige Leistungsfähigkeit, immer mehr nützliche Funktionen während eines physikalischen Experimentes übernehmen.

Zu solchen Funktionen gehören beispielsweise:

1. Die Simulation von physikalischen Experimenten
2. Das Aufnehmen von Messdaten während des Verlaufs eines Experimentes
3. Das numerische Lösen von theoretischen Sachverhalten eines Experimentes
4. Oder das Umsetzen von Regelprozessen durch ein automatisiertes Verfahren

Der vierte Punkt der obigen Funktionen soll Kernbestandteil dieser Arbeit sein.

Typische Regelungsprozesse oder Regelkreise finden sich heutzutage in allen möglichen Bereichen. Ein Beispiel ist das **A**nti**b**lockier**s**ystem (kurz ABS) in der Fahrzeugtechnik. Das ABS verhindert das Blockieren der Fahrzeugräder bei Vollbremsungen.
Durch ein **E**ingebettetes **S**ystem (kurz ES), innerhalb des Fahrzeugs, wird die Bremskraft einzelner Radbremsen verringert, sobald sich das Rad nicht weiterdreht. Theoretisch existiert für jede Geschwindigkeit eines Fahrzeugs eine optimale Bremskraft, welche eine maximale Bremsbeschleunigung des Fahrzeugs hervorruft, wobei sich die Räder gerade noch weiterdrehen. Da diese optimale Bremskraft von der momentanen Bewegungsgeschwindigkeit und der Bodenbeschaffenheit abhängt, ist ein Regelkreis notwendig, der permanent diese optimale Bremskraft einstellt.

Ein anderes Beispiel für einen Regelkreis ist die Regelung der Lüftergeschwindigkeiten als Aktivkühler eines Prozessors. Es existiert eine optimale Einstellung der Lüfterumdrehungen pro Minute, welche einen bestimmten Luftvolumendurchsatz pro Minute erzeugt und so den Prozessor bei momentaner Abwärmeleistung auf einer bestimmten Arbeitstemperatur hält.

1.1 Motivation

Um das Erstellen und Entwickeln von Regelkreisen mit dem PC zu erlernen, soll im Rahmen eines physikalischen Praktikumversuches ein konkretes Experiment aufgebaut und durchgeführt werden. Innerhalb des besagten Experimentes bedarf es einer Regelung, welche mit dem PC vorgenommen werden soll.
Es wird also ein PC, samt Hardware und Softwareausstattung, benötigt, der in der Lage ist Regelschaltungen in Echtzeit umzusetzen und dadurch in das Experiment eingebunden werden kann.

1.2 Inhalt der Arbeit

Diese Arbeit beschäftigt sich mit dem Vorbereiten eines solchen PC, welcher für die Umsetzung einer Regelschaltung für ein physikalisches Experiment genutzt werden kann. Dazu werden „Gentoo“-Linux, RTAI, Scilab/Scicos und Comedi verwendet. Im Folgenden wird der betroffene PC als „Experiment-PC“ bezeichnet.
Das physikalische Experiment wurde speziell präpariert, so dass eine Regelung ausgeführt werden muss. Nähere Informationen zum Experiment finden sich im Kapitel 2.

Im Verlauf der Arbeit werden Hardware- sowie Software-Ansprüche und deren Anpassungen besprochen, so dass alle Voraussetzungen des Experiment-PC, zur Umsetzung eines zuverlässigen Regelprozesses, erfüllt werden.
Aus Gründen, welche in dieser Arbeit erläutert werden, wurde der Experiment-PC mit der Open-Source Linux-Distribution namens „Gentoo“ ausgestattet. Zusätzlich verfügt der Experiment-PC über eine Mess- und Steuerkarte, welche es dem Experiment-PC erlaubt Signale aus dem Experiment aufzunehmen und eine entsprechende Regelung, durch die Ausgabe von Signalen, vorzunehmen.

Nähere Erläuterungen zum Aufbau des Experiment-PC erfolgt in Kapitelpunkt 2.1.4.

1.2.1 Gliederung der schriftlichen Ausarbeitung

In **Kapitel 2** dieser Arbeit wird das Experiment, sowie der experimentelle Aufbau erläutert. Dadurch wird erkenntlich wo während des Experimentes eine Regelung vorgenommen werden muss.

Zusätzlich wird der Experiment-PC mit seiner Hardwareausstattung vorgestellt, mit dem die Regelung durchgeführt werden soll.

Durch das **Kapitel 3** werden alle Grundlagen aufbereitet, welche benötigt werden um dem weiteren fachlichen Inhalt der Arbeit folgen zu können und um dargestellte Probleme sowie gewählte Lösungen nachvollziehbar zu machen.
Es werden Schwerpunkte auf allgemeine Möglichkeiten der Realisierung von PID-Regelungen im analogen und digitalen Bereich gelegt. Zusätzlich wird erklärt, was unter dem Begriff Echtzeit verstanden wird und welchen Nutzen ein Echtzeit-System zur Umsetzung der Ziele dieser Arbeit hat.

In **Kapitel 4** werden Softwarepakete und Betriebssysteme vorgestellt, welche Möglichkeiten bieten einen Regelkreis in Echtzeit durchzuführen. Es wird ebenfalls erläutert in wie weit die Software LabVIEW für Echtzeit-Regelungen verwendet werden kann.
Des weiteren werden in diesem Kapitel die grundlegenden Funktionsweisen der RTAI-Erweitung für ein Linux erklärt. Dies ist wichtig, da das RTAI den gewählten Weg zur Erreichung der Ziele in dieser Arbeit darstellt.

Kapitel 5 erklärt genau den Ansatz, der mit dieser Arbeit vorgenommen wurde, um ein Experiment-PC zu erhalten mit dem Regelungen in Echtzeit ausgeführt werden können. Es wird dabei auf die verwendeten Softwarepakete und deren Zusammenspiel eingegangen.

Das **Kapitel 6** soll die Schritte zur Vorbereitung des Experiment-PC aufzeigen und erläutern. Es wird zuerst erklärt wie das „Gentoo“-Linux-Grundsystem auf dem Experiment-PC installiert wurde. Dann wird eine Erläuterung der RTAI-Erweiterung des Grundsystems erfolgen.
Da diese Ausführungen auf dem aktuellen Experiment-PC umgesetzt wurden, wird in diesem Kapitel konkret auf den Experiment-PC Bezug genommen.
Im letzten Kapitelpunkt des Kapitels werden alle angetroffnen Probleme, während der Einrichtung des Experiment-PC, zusammenfassend erläutert. Zusätzlich wird die Lösung der Probleme und die Entscheidung zur Lösung diskutiert.

In **Kapitel 7** werden alle Ergebnisse der Arbeit zusammengefasst. Es wird weiter erläutert in wie weit der eingerichtete Experiment-PC theoretisch in der Lage ist seinen Zweck im Aufbau des Experimentes zu erfüllen.
Um den Zustand des Experiment-PC zu erläutern werden in diesem Kapitel Tests durchgeführt, welche das Ausmaß der Echtzeitfähigkeit des Experiment-PC verdeutlichen.

Kapitel 8 erläutert alle möglichen Ansätze, um die Regelung des Experiments mit dem entstandenen Experiment-PC durchzuführen. Es soll auf Grenzen der praktischen Umsetzung hingewiesen und Ansätze zu möglichen Updates auf dem Experiment-PC aufgezeigt werden.
Als weiterer Ausblick wird ein Weg besprochen, mit dem die Regelung für das Experiment in

Hardware geschehen kann. Dieser Weg ist theoretisch mit dem, in dieser Arbeit vorbereiteten, Experiment-PC problemlos möglich.

Kapitel 2

Das Experiment

Um eine digitale Regelung mit dem Experiment-PC praktisch durchzuführen, wurde ein physikalisches Experiment vorbereitet, welches mehrerer optionaler Regelungen bedarf.
Grundsätzlich handelt es sich bei dem Experiment um ein Fabry-Pérot-Interferenz-Experiment.

2.1 Materialien

Es werden im Folgenden nur die Experimentmaterialien beschrieben, die benötigt werden um das Grundprinzip des Experimentes zu verdeutlichen.

Laser

Es stehen 2 Laser zur Verfügung, von denen einer ein **He**lium-**Ne**on **L**aser (kurz He-Ne-L) und einer ein **H**alb**l**eiter **L**aser (kurz HL-L) ist.

Helium-Neon Laser :

- Polytec Typ PL710
- Optical output power 2,5 mW
- Wellenlänge 633 nm

Halbleiter Laser (HL6722G):

- AlGaInP Laser Diode

- Wellenlänge von 660 nm bis 680 nm (abhängig von Strom und Temperatur)
- Optical output power 5 mW

Die Wellenlänge des HL-L ist von der Temperatur der Laserdiode und vom Laserstrom abhängig.

Fabry-Pérot-Resonator

Ein weiteres wichtiges Element ist der Fabry-Pérot-Resonator mit Piezo-Element. Dieser Resonator wird durch zwei parallele Spiegel gebildet, wobei der zweite Spiegel durch ein Piezo-Element mit seiner Halterung verbunden ist. Über das Piezo-Element kann die Lage des zweiten Spiegels extern beeinflusst werden. Dadurch ergeben sich einstellbare Resonatorlängen (l).

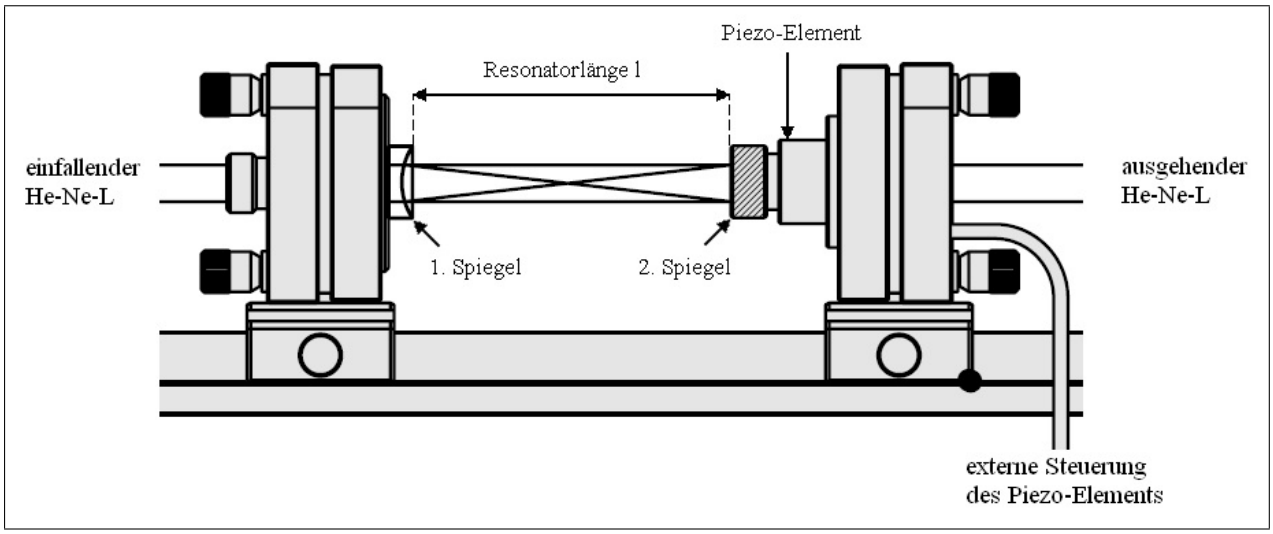

Abbildung 2.1: Der Fabry-Pérot-Resonator innerhalb des Experimentes [fabres]

Abbildung 2.1 zeigt schematisch den Aufbau des Fabry-Pérot-Resonators und seine Wirkungsweise im Experiment. Über die externe Steuerung des Piezo-Elements können Hochspannungen auf den Piezoaktor übertragen werden. Diese Hochspannungen resultieren in eine Längenänderung des Piezo-Elements und damit der Resonatorlänge.

Piezo-Element

Das Piezo-Element besteht aus mehreren Piezoaktoren, welche getrennt angesteuert werden können, um ein Kippen des 2. Reflektionsspiegels zu erreichen. Bei der praktischen Umsetzung des Experimentes werden jedoch alle Piezoaktoren gleichmäßig angesteuert, da das Spiegelsystem des

Resonators selbstfokussierend ist und so keine Veränderung des Resonatorstrahlenganges notwendig ist.

Detektor

Am Ausgang des Fabry-Pérot-Resonator entsteht ein Interferenzbild, das mit einem Detektor aufgenommen werden soll. Der Detektor besteht prinzipiell aus einer Photodiode, welche die Lichtintensität des Interferenzmusterverlaufs in elektrische Signale umwandelt. Die entstehenden elektrischen Signale können beliebig weiter verarbeitet werden.

2.2 Aufbau und Durchführung

Teilexperiment 1

Ein vollständiger Aufbau des Experimentes mit Teilexperiment 1 sieht wie folgt aus:

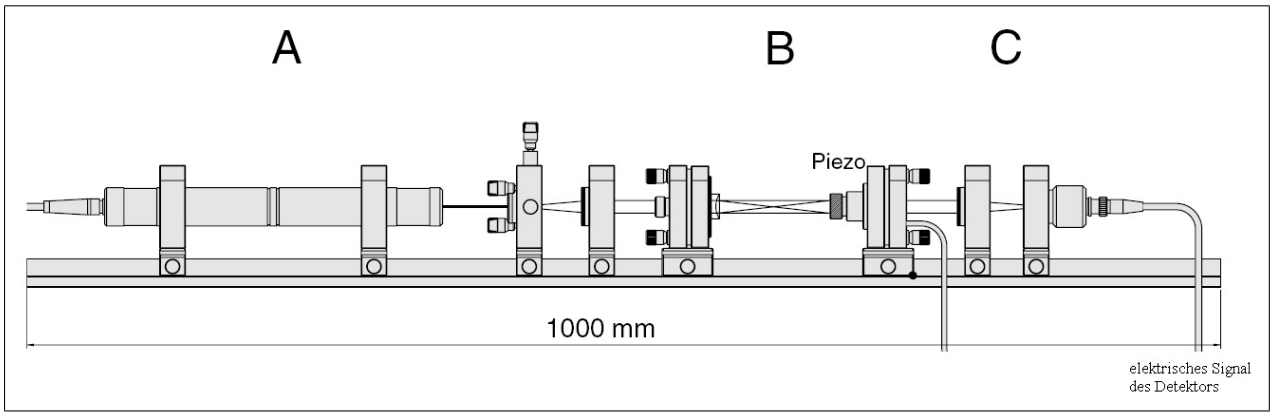

Abbildung 2.2: schematischer Experimentaufbau des Teilexperiments 1 [fabres]

In Abbildung 2.2 zeigt die Sektion A den He-Ne-L, dessen Strahl in die Sektion B, den Fabry-Pérot-Resonator, eintritt und dort mit sich selbst überlagert wird. Das entstehende Interferenzbild tritt in die Sektion C ein und wird dort mit dem Photodioden-Detektor untersucht.
Es existiert nun in der Abbildung 2.2 ein relevantes Eingangssignal für den Experiment-PC. Dies wäre die Lichtintensitätsmessung des Detektors. Der Detektor wird auf den Anstieg einer konstruktiven Interferenz fokussiert. Der Signalverlauf des Detektors gibt somit Aufschluss über die Bewegung des Interferenzbildes.
Das Ausgangssignal, welches von Experiment-PC erzeugt werden muss, ist die externe Steuerung des Piezo-Elements, in Abbildung 2.1 bezeichnet.

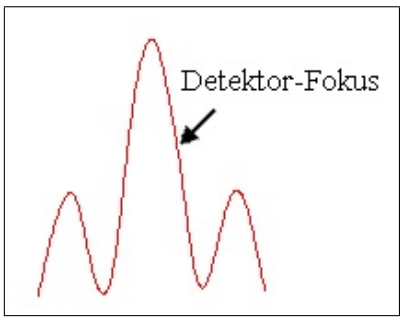

Abbildung 2.3: Fokus des Detektors auf der konstruktiven Interferenz

Abbildung 2.3 zeigt den Fokus des Detektors. Bei Detektierung einer ansteigenden oder abfallenden Lichtintensität kann, je nach Orientierung, auf eine Verlängerung oder Verkürzung der Resonatorlänge geschlossen werden.
Würde der Fokus des Detektors auf einem Maximum oder Minimum des Interferenzbildes gerichtet sein, so könnte zwar eine Änderung der Intensität festgestellt werden, diese Änderung wäre jedoch in beide Richtungen identisch und könnte so nicht zur Feststellung der Änderungsrichtung der Resonatorlänge verwendet werden.

Teilexperiment 2

Ein zweites Teilexperiment nutzt den zusätzlichen HL-L. Der HL-L wird in den Strahlengang des He-Ne-L eingebracht und erzeugt mit ihm ein Interferenzbild. Das entstehende Interferenzbild wird ebenfalls detektiert und wird auf Änderungen untersucht.

Ändert sich nun das Interferenzbild, so sind diese Änderungen, bei kompensierten Resonatorlängenänderungen, auf die Wellenlängenänderung des HL-L zurückzuführen. Die Wellenlänge des HL-L ändert sich, da diese temperatur- und stromabhängig ist. Während des Betriebes des HL-L kann die Temperatur nicht konstant gehalten werden. Eine Temperaturänderung des HL-L führt also zu einer Verschiebung des Interferenzbildes.

2.3 Die Regelkreise

Die Teilexperimente in Kapitelpunkt 2.2 wurden so manipuliert, dass eine durch den Experiment-PC gesteuerte Regelung notwendig wird.
Ein Beispiel für diese Änderung ist die Ansteuerung des HL-L. Normalerweise wird der HL-L durch seine Steuerstation so angesteuert, dass sein Betriebsstrom selbstständig für eine konstante Wellenlänge angepasst wird. Diese Ansteuerung kann aber auch manuell durch ein äußeres Signal vorgenommen werden. Dieses äußere Signal soll das Regelsignal des Experiment-PC sein.

Teilexperiment 1

Um nun das Interferenzbild des Teilexperimentes 1 konstant zu halten, soll eine Regelung mit dem Experiment-PC vorgenommen werden. Durch den Experiment-PC muss eine entsprechende Ansteuerung des Piezo-Elements stattfinden, welche eine konstante Resonatorlänge bewirkt. Diese Ansteuerung soll das Interferenzbild konstant halten.
Störeinflüsse auf die Resonatorlänge sind alle Einflüsse auf den Experimentaufbau, wie die Halterungen oder das Piezo-Element.

Unter anderem sind solche konkreten Störeinflüsse beispielsweise:

- Erschütterungen des Experimentaufbaus
- Sprache in der Umgebung des Experimentaufbaus
- Temperaturänderungen und dadurch Längenänderungen aller Halterungen und des Piezo-Elements

Als Eingangssignal für den Experiment-PC dient das elektrische Signal des Detektors. Das Eingangssignal gibt darüber Auskunft, ob sich die Resonatorlänge geändert hat und wenn ja, in welche Richtung.
Als Regelkreis soll nun mit dem Experiment-PC eine geeignete PID-Regelung programmiert werden, welche eine Gegensteuerung mit dem Piezo-Elements vornimmt und so die Resonatorlänge konstant hält.

Teilexperiment 2

Bei diesem Teilexperiment soll ebenfalls ein konstantes Interferenzbild erzeugt werden. Hierzu muss der Laserstrom des HL-L jederzeit so nachgeregelt werden, dass der HL-L ein Laserlicht mit konstanter Wellenlänge emittiert.

Störeinflüsse bei diesem Teilexperiment können unter anderem sein:

- Stromschwankungen durch die HL-L Steuerstation
- Widerstandsänderungen der HL-L Bauelemente durch Erwärmung
- Erwärmung der Halbleiterdiode und die damit verbundene Änderung der Wellenlänge

Der Regelkreis ist mit dem Experiment-PC ebenfalls durch eine PID-Regelung möglich.

2.4 Der Experiment-PC

Bei dem Experiment-PC handelt es sich um einen PC mit folgender Hardwareausstattung:

- AMD Athlon^{TM} XP 2400+
- Mainboard Chipsatz von VIA
- 1 GByte RAM
- NVIDIA Geforce FX 5200 Grafikkarte
- Mess- und Steuerkarte ME-4660i von der Firma Meilhaus

Allgemeine Hardware

Die Wahl der Geschwindigkeit des Experiment-PC ist von großer Bedeutung, da der Experiment-PC in jeder Regelperiode die Berechnungen des Regelalgorithmus ausführt. Je nachdem wie schnell diese Berechnungen abgeschlossen sind könnte danach die nächste Regelperiode beginnen.
Mehr Informationen zum Thema Berechnungszeit und Latenzzeit unter Echtzeitbedingungen im Kapitelpunkt 3.2.

Mess- und Steuerkarte

Die Mess- und Steuerkarte ME-4660i von Meilhaus Electronic ist eine PC-Karte für den PCI-Steckplatz. Mit ihr ist es möglich mit elektrischen Bauteilen außerhalb des Experiment-PC zu interagieren. Es können also analoge Signale von Sensoren in den Experiment-PC eingelesen oder analoge Regelungen am Experiment vorgenommen werden.

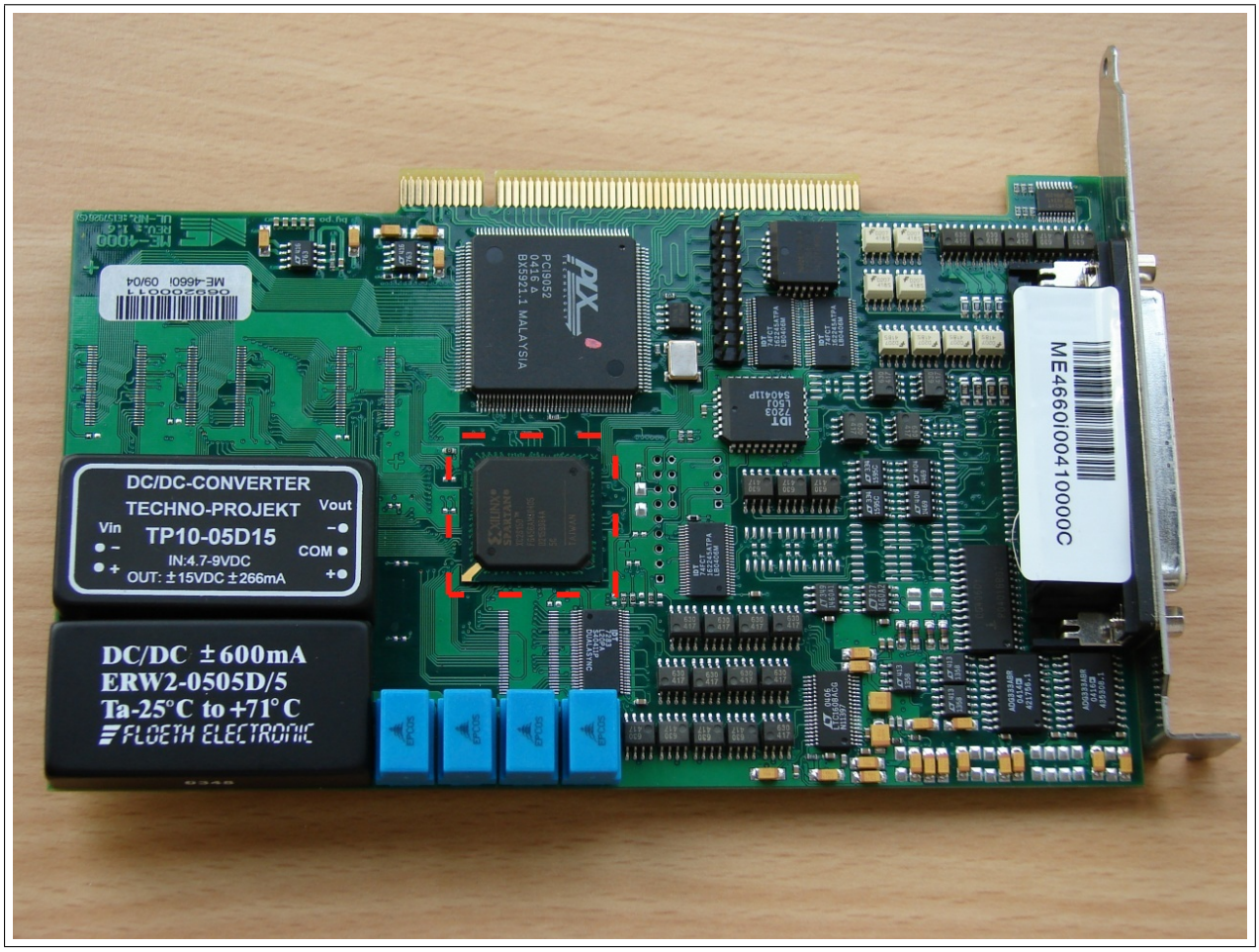

Abbildung 2.4: Die ME-4660i Mess- und Steuerkarte von Meilhaus

Die Abbildung 2.4 zeigt die ME-4660i Mess- und Steuerkarte für den PCI-Steckplatz. Der rote Rahmen in Abbildung 2.4 markiert einen Field Programmable Gate Array (kurz FPGA), der sich auf der ME-4660i befindet. Dieser FPGA wird im Kapitelpunkt 8.2.1 näher betrachtet.

Die ME-4660i verfügt über:

- 16 analoge Eingänge (subdevice 0) mit je 16 Bit Auflösung. Mit diesen Eingängen werden analoge Eingangssignale über einen A/D-Wandler in den Experiment-PC eingelesen. Die 16 Bit Auflösung können gleichmäßig auf eines der Spannungsintervalle ([0 V, 2.5 V]; [0 V, 10 V]; [-2.5 V, 2.5 V]; [-10 V, 10 V]) aufgeteilt werden.
- 2 analoge Ausgänge (subdevice 1) mit je 16 Bit Auflösung. Über diese Ausgänge können digitale Signale über einen D/A-Wandler als analoges Signal ausgegeben werden. Die 16 Bit Auflösung sind nur auf das Spannungsintervall [-10 V, 10 V] anwendbar.
- Ebenfalls verfügt die ME-4660i über 32 bidirektionale I/O Ports und 3 Counter, welche in dieser Arbeit nicht verwendet wurden.

- Es befindet sich ein rekonfigurierbarer FPGA („Spartan II XC2S150“) von der Firma Xilinx auf der ME-4660i.

Betriebssystem

Als Betriebssystem wurde in dieser Arbeit „Linux“ benutzt, da für Linux komfortable Open-Source Lösungen im Bereich Echtzeit-Betriebssysteme angeboten werden.
Weiter wurde die Linux Distribution „Gentoo“ in dieser Arbeit verwendet, da Gentoo durch den Benutzer optimal auf ein Computersystem angepasst werden kann, um so die besten Voraussetzungen für geringe Latenzzeiten zu erhalten.

Besonderheiten von Gentoo-Linux

Die Linux Distribution „Gentoo“ ist eine quelltextbasierende Distribution. Es wurden alle, in dieser Arbeit, zu installierenden Softwarepakete als Quelltext aus dem Internet heruntergeladen und speziell für den Experiment-PC kompiliert. Zur Kompilierung werden in Gentoo Use-Flags verwendet, die Auskunft über die Hardware des Experiment-PC geben.
Die Distribution Gentoo verfügt über zwei charakteristische Elemente, den Portage-Tree (deut.: Portagebaum) und das „emerge“-Tool.

Der **Portage-Tree** ist eine Ansammlung von sogenannten „ebuilds“. Ein ebuild ist eine Zusammenstellung von Instruktionen, um ein bestimmtes Paket zu kompilieren, zu installieren und zu konfigurieren. In ebuilds sind weitere Informationen wie Paketabhängigkeiten, Homepages für den Quellcode und verfügbarer Versionen eines Paketes enthalten. Der Portage-Tree wird bei einer Gentoo Distribution im Verzeichnis „`/usr/portage`“ abgelegt.
Der Gentoo-Benutzer kann also zwischen vielen Versionen wählen, wenn er ein Paket installieren möchte.

Um alle ebuilds im Portage-Tree verwalten, ausführen oder updaten zu können wird das „**emerge**“-Tool verwendet. Mit diesem Tool ist es möglich eine benutzerfreundliche Installation aller Gentoo-Pakete vorzunehmen. Das „emerge“-Tool liest die entsprechenden ebuilds und lädt den Quellcode des Paketes herunter und kompiliert es mit den Computer eigenen Use-Flags. Eine Deinstallation, von Paketen, ist ebenfalls mit dem „emerge“-Tool möglich.

Kapitel 3

Grundlagen

Regelprozesse werden heute in der Industrie und Forschung immer häufiger mit Hilfe der digitalen Technik vorgenommen. Als Stützpunkt solcher digitalen Regelungen gelten ES oder vollständige Computer wie PCs.
Bei ES- oder computergestützter Regelung handelt es sich um eine digitale, zeitdiskrete Regelung. Eine digitale Regelung führt somit eine Signalquantisierung ein und besitzt eine definierte Abtastperiodendauer oder Regelperiodendauer.
Im Gegensatz zur digitalen Regelung steht die analoge, zeitkontinuierliche Regelung, welche durch die Analogtechnik realisiert werden kann. Analoge Regelungen haben die Eigenschaft einen stetigen Regelverlauf zu besitzen, bei digitalen Regelungen ist dieser Verlauf dagegen sprunghaft.

Der Vorteil eines Computers, zum Umsetzen von Regelprozessen, ist die enorme Flexibilität der Regelung. Bei einem Computer findet der Regelverlauf nach den Vorschriften eines Programms statt und kann jederzeit verändert werden. Dadurch ergeben sich schnelle Entwicklungs- und Wartungszeiten. Die technische Wartung von Regelungen wird dadurch erleichtert, dass lediglich das Regelprogramm ausgetauscht werden muss ohne Änderungen an analogen Schaltungen oder Platinen vorzunehmen.

3.1 Regelung

Durch den Vorgang der Regelung soll eine bestimmte physikalische Größe auf einem bestimmten Wert gehalten werden. Diese physikalische Größe wird „Regelgröße“ genannt und der zu haltende Wert wird „Sollwert“genannt. Die Differenz zwischen Regelgröße und Sollwert wird „Regeldifferenz“ genannt.
Es wird explizit zwischen Regelung und Steuerung unterschieden.

Eine Steuerung nimmt Änderungen der Regelgröße vor ohne deren tatsächliche Übereinstimmung mit dem Sollwert zu überprüfen oder die Abweichung vom Sollwert in den Steuerprozess einzubinden.

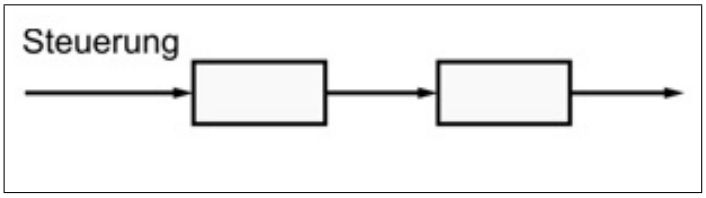

Abbildung 3.1: schematischer Ablauf einer Steuerung

Die Abbildung 3.1 zeigt die offene Wirkungskette einer Steuerung. Eine Steuerung berücksichtigt nur die, in die Wirkungskette, eingehende Messgröße. Andere Störgrößen werden nicht berücksichtigt und nicht kompensiert.

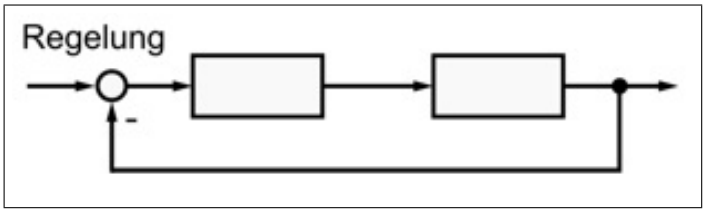

Abbildung 3.2: schematischer Ablauf einer Regelung

In Abbildung 3.2 ist schematisch der Ablauf einer Regelung dargestellt. Die Regelgröße wird auf den Eingang der Regelkette rückgekoppelt. An dem Kreis, in Abbildung 3.2, mit dem Minuszeichen ist zu erkennen, dass die Regeldifferenz als Eingabe der Regelkette genutzt wird.

Regelung

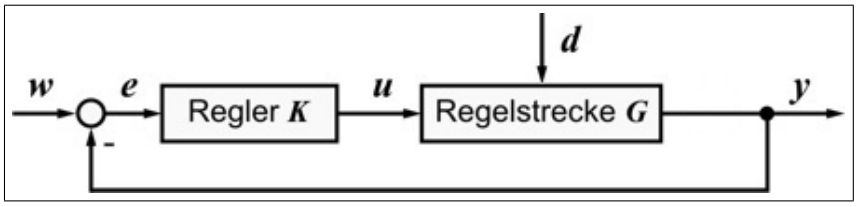

Abbildung 3.3: schematischer Ablauf einer Regelung

In Abbildung 3.3 ist der Ablauf einer Regelung detaillierter dargestellt. Aus Regelgröße (y) und dem Sollwert (w) wird die Regeldifferenz (e) bestimmt. Die Regeldifferenz wird vom Regler (K) benutzt um einen neuen Stellwert (u) zu berechnen. Dieser Stellwert wirkt auf die Regelstrecke (G) ein und bewirkt, unter Einfluss der Störgrößen (d), eine neue Regelgröße.

Eine Regelung nimmt eine Einstellung der Regelgröße vor und misst die Abweichung der Regelgröße vom Sollwert. Diese Abweichung hat Einfluss auf den Regelprozess oder Regelalgorithmus. Es findet also eine Rückkopplung der Regelgröße durch die Regeldifferenz in den Regelprozess statt.

Eine weitere wichtige Größe sind die „Störgrößen“. Die Störgrößen bewirken eine nicht beeinflussbare Änderung der Regelgröße. Dadurch wird eine permanente Regelung notwendig, um die sich ändernden Störgrößen auszugleichen. Eine Regelung bekämpft den Einfluss aller Störgrößen im Regelkreis.
Störgrößen sind beispielsweise die Widerstandsänderungen bei sich ändernden Temperaturen, da diese Einfluss auf den Betriebsstrom haben. Oder Änderungen der Netzspannung bei Regelung der Umdrehungszahl eines Elektromotors.
Währen bei einer Regelung keine Störgrößen vorhanden oder würde sich der Sollwert nicht ändern, so wäre auch keine Regelung notwendig, da der einmal eingestellte Regelwert auf dem Sollwert verharren würde.

Beispiel einer Regelung

Ein einfaches Beispiel wäre ein Tauchsieder in einem Wasserbecher. Es soll nun das, im Becher enthaltene, Wasser auf konstanten 70°C gehalten werden, dies wäre der Sollwert. Die Regelgröße ist die Momentantemperatur des Wassers. Der einzige Stellwert, welcher in diesem Beispiel zur Verfügung steht, ist der Betriebsstrom des Tauchsieders. Der Betriebstrom muss nun mit einem Regelkreis so eingestellt werden, dass die Wassertemperatur den Sollwert erreicht. Damit für den Regelvorgang eine Regeldifferenz ermittelt werden kann, könnte sich ein Widerstandsthermometer im Wasserbecher befinden.

Aufgrund der zahlreichen Störgrößen welche auf die Temperatur des Wassers einwirken ist es unter Umständen nicht möglich festzulegen, wie viel Strom durch den Tauchsieder fließen muss um das Wasser auf einer bestimmten Temperatur zu halten. Könnte diese Funktion $T = f(I)$ angegeben werden so würde eine Regelung zu einer Steuerung entarten.
Eine mögliche Störgröße für das Tauchsiederszenario wäre beispielsweise die Temperatur und Wärmeleitfähigkeit der Becherumgebung. Temperatur und Wärmeleitfähigkeit der Umgebung haben Einfluss auf die Wärmeverlustleistung des Wassers im Becher. Eine erhöhte Wärmverlustleistung muss mit einer höheren Leistung des Tauchsieders ausgeglichen werden, um das Wasser auf einer konstanten Temperatur zu halten. Da die Wärmeverlustleistung nicht für jede Umgebung-Temperatur-Kombination bekannt sein kann muss eine Regelung vorgenommen werden.

3.1.1 Analoge Regelung

Bei der analogen Regelung handelt es sich um eine zeitkontinuierliche und stetige Änderung des Stellwertes und damit der Regelgröße. Analoge Regelungen werden mit passiven elektrischen Bauelementen wie Kondensatoren oder Spulen und aktiv mit Operationsverstärkern durchgeführt.

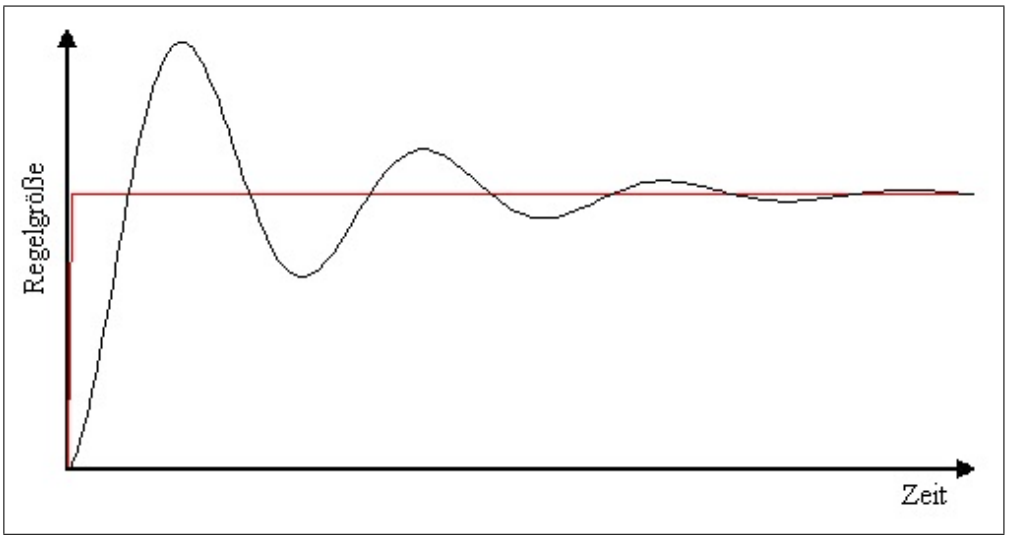

Abbildung 3.4: Regelverlauf (schwarz) einer analogen Regelung bei Sprungänderung des Sollwertes (rot)

Abbildung 3.4 zeigt ein Beispiel einer analogen Regelung. Der Verlauf der Regelung ist zeitkontinuierlich und stetig. Mit zunehmender Zeit nähert sich die Regelgröße dem Sollwert an.

Vorteile der analogen Regelung sind:

- Theoretisch unbegrenzte Auflösung bei der Messung und Einstellung der Regelgröße und des Sollwertes
- Hohe Frequenzen von bis zu mehreren GHz sind mit Operationsverstärkern möglich.
- Analoge Regelkreise sind für geringe Anforderungen kostengünstiger

Nachteile der analogen Regelung sind:

- Regelprozesse mit vielen Mess- und Stellgliedern werden mit analogen Bauelementen sehr kompliziert und umfangreich.
- Analoge Regelkreise sind weniger flexibel, wenn es darum geht vergangene Messwerte zu verarbeiten.
- Wurde der Regelkreis einmal erstellt so ist er nur schwer zu ändern oder auszutauschen.

- Mit einem analogen Regelkreis kann nur schwer der Verlauf der Regelung aufgezeichnet oder gespeichert werden.

3.1.2 Digitale Regelung

Bei digitalen Regelungen handelt es sich um zeitdiskrete und unstetige Änderungen der Regelgröße. Die Änderungen können mit einem ES oder PC-System durchgeführt werden.

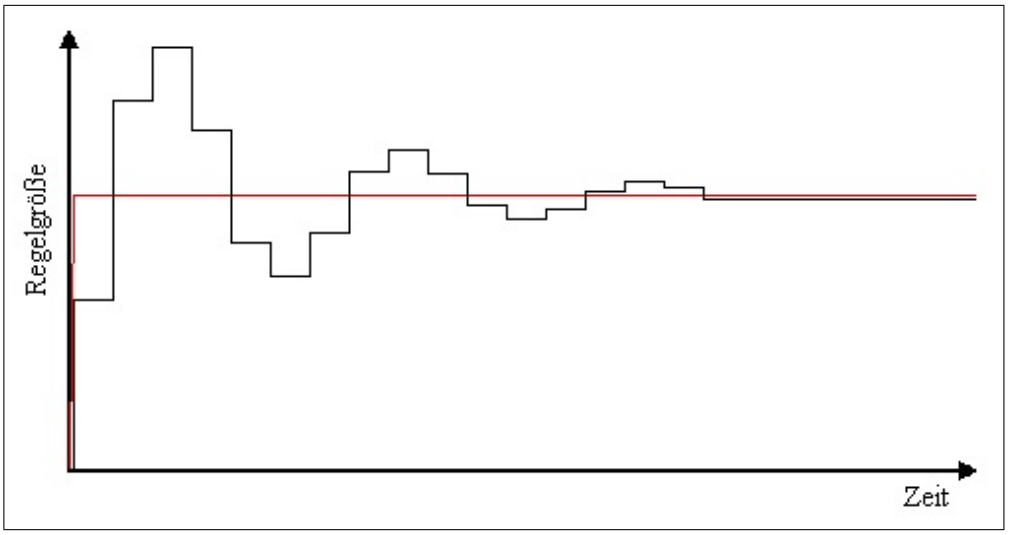

Abbildung 3.5: Regelverlauf (schwarz) einer digitalen Regelung bei Sprungänderung des Sollwertes (rot)

Abbildung 3.5 zeigt ein Beispiel einer digitalen Regelung. Der Verlauf der Regelung ist zeitdiskret und unstetig. Mit zunehmender Zeit nähert sich die Regelgröße dem Sollwert an. Allerdings ist eine bleibende Abweichung zwischen Regelgröße und Sollwert zu erkennen. Dies kann eine Folge der Signalquantisierung sein, die bei jeder digitalen Regelung auftritt.

Die unstetige Natur der digitalen Regelung hat ihren Ursprung in der Digitaltechnik. Computer können für Messwerte oder Regelgrößen nur diskrete Werte messen oder einstellen. Dies führt zu einer permanenten Ungenauigkeit der Regelgröße gegenüber dem Sollwert. Diese Ungenauigkeit muss, durch Vergrößerung der Auflösung, soweit verringert werden, bis sie für den Regelprozess irrelevant wird. Die zeitdiskrete Eigenschaft der digitalen Regelung bedeutet, dass ein eingestellter Regelwert für eine bestimmte Regelperiodedauer konstant anliegt. Erst wenn eine neue Periode der digitalen Regelung beginnt, kann eine neue Regelgröße eingestellt werden.
Der Regelprozess an sich wird, bei digitalen Regelungen, durch den Ablauf eines Computerprogramms bestimmt. Das sogenannte Regelprogramm beinhaltet den Regelalgorithmus, der für jede Regelperiode einen neuen Stellwert berechnet und damit eine neue Regelgröße erzielt.
Durch die Verwendung eines Regelprogramms ergibt sich eine enorme Flexibilität der digitalen

Regelung. Alles was praktisch programmiert werden kann, könnte theoretisch als Regelung dienen.

Vorteile der digitalen Regelung sind:

- Es können sehr viele Mess- und Stellglieder verarbeitet und eingestellt werden.
- Der Regelprozess ist durch das Regelprogramm schnell austauschbar oder veränderbar.
- Das Speichern von Messwerten und Auswerten von gespeicherten Messwerten kann unkompliziert vorgenommen werden.
- Es müssen nicht bekannte analoge Regelverfahren verwendet werden. Es können auch vollkommen neue Regelalgorithmen entworfen werden.

Nachteile von digitalen Regelungen sind:

- Digitale Regelungen besitzen ein computerabhängiges Auslösungsvermögen.
- Die Dauer der Regelperiode ist stark Computerabhängig und ist selbst mit modernen PCs bestenfalls im MHz-Bereich möglich.
- Das Regelprogramm muss die Echtzeitbedingung auf dem verwendeten Computer erfüllen, sonst wäre es theoretisch möglich, dass selbst der schnellste Regelalgorithmus unbrauchbar würde. Nähere Informationen erfolgen im Kapitelpunkt 3.2.

3.1.2.1 Entscheidung zur digitalen Regelung

Ein wichtiges Kriterium der digitalen, computergesteuerten Regelung ist die Echtzeitbedingung. Kann diese Bedingung in Kombination eines Regelalgorithmus mit einem bestimmten Computersystem nicht erfüllt werden, so ist diese Kombination für die Regelung unbrauchbar.

3.1.3 PID-Regelung

Der Begriff PID-Regelung steht für **P**roportional-**D**ifferential-**I**ntegral-Regelung (kurz PID). PID-Regelungen sind Regelprozesse, welche in ihrem Verlauf einen proportionalen, differentialen und integralen Anteil besitzen.
Der PID-Regler ist ein häufig verwendeter Regler, um eine Regelgröße auf einen bestimmten Sollwert zu halten. Die PID-Regelung kann mit analogen Bauelementen realisiert werden, dies ist jedoch nicht zwingend. Egal wie der Regelprozess realisiert wird, wenn er einen proportionalen, differentialen und integralen Anteil besitzt, so wird er PID-Regler genannt.

P-Anteil

Der proportionale Anteil kann auch einzeln Proportional-Regler (kurz P-Regler) genannt werden.

Die Übertragungsfunktion eines P-Reglers ist:

$$\boxed{y(t) = p \cdot e(t)}$$

$y(t)$ ist der zeitliche Verlauf der Regelgröße
$e(t)$ ist der zeitliche Verlauf der Regeldifferenz
p ist der charakteristische Proportionalverstärkungsfaktor des P-Reglers

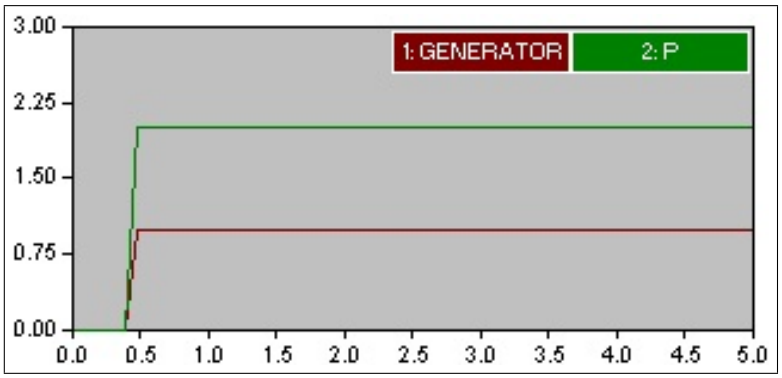

Abbildung 3.6: Sprungantwort eines P-Reglers

In der Abbildung 3.6 ist die Sprungantwort eines P-Reglers mit $p = 2$ zu erkennen. Der Vorteil des P-Reglers ist, dass er sofort auf eine Änderung der Regeldifferenz reagiert. Der P-Regler besitzt eine permanente Regeldifferenz und tendiert bei großen p zum Schwingen.

I-Anteil

Der integrale Anteil kann auch einzeln **I**ntegral-Regler (kurz I-Regler) genannt werden. Der I-Regler bestimmt seine Regelgröße aus dem Integral über den zeitlichen Verlauf der Regeldifferenz.

Die Übertragungsfunktion eines I-Reglers ist:

$$\boxed{y(t) = i \cdot \int_0^t e(\tau)\, d\tau}$$

i ist der charakteristische Integralverstärkungsfaktor des I-Reglers

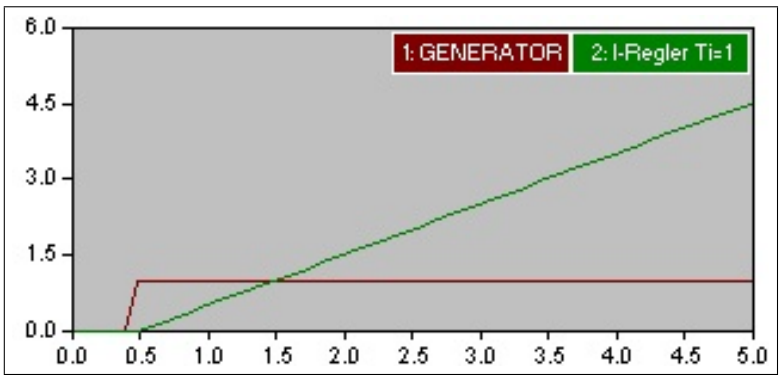

Abbildung 3.7: Sprungantwort eines I-Reglers

Die Abbildung 3.7 zeigt die Sprungantwort eines I-Reglers mit $i = 1$. Der I-Anteil eines Regelprozesses ist ausschlaggebend für die Genauigkeit des Reglers. Sollte der Regelverlauf des I-Reglers den Sollwert passieren, so würde sich das Vorzeichen der Regeldifferenz umkehren und der Regelverlauf würde sich an den Sollwert angleichen.

Sollte ein I-Regler mit einem Computer umgesetzt werden, so ist es nicht möglich eine perfekte Integration zu benutzen. Stattdessen werden die Regelabweichungen aller vorangegangenen Regelperioden summiert und so der Effekt einer Integration erzeugt. Die Übertragungsfunktion ändert sich dann zu:

$$\boxed{y(t) = i \cdot \sum_{x=0}^{n} e(t_x)}$$

Dabei ist zu beachten, dass in einem Computer nur diskrete Zeitschritte existieren. Diese Zeitschritte werden in der Übertragungsfunktion mit t_x bezeichnet.

D-Anteil

Der differentielle Anteil eines Reglers kann auch einzeln **D**ifferential-Regler (kurz D-Regler) genannt werden. Der D-Regler bestimmt seine Regelgröße aus der zeitlichen Ableitung des Regeldifferenzverlaufs. Durch den D-Anteil eines Reglers wird also nur ein Regelung ungleich Null vorgenommen, wenn sich die Regeldifferenz ändert.

Die Übertragungsfunktion eines D-Reglers lässt sich wie folgt formulieren:

$$\boxed{y(t) = d \cdot \frac{de(t)}{dt}} \qquad d \quad \text{ist der charakteristischer Differentialverstärkungsfaktor des D-Reglers}$$

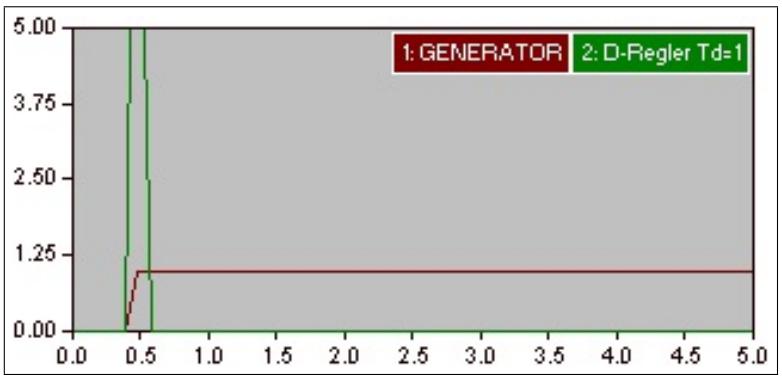

Abbildung 3.8: Sprungantwort eines D-Regler

Die Abbildung 3.8 zeigt die Sprungantwort eines D-Reglers mit $d = 1$. Sollte ein D-Regler mit einem Computer umgesetzt werden, so ist es nicht möglich eine perfekte Ableitung zu benutzen. Stattdessen wird die Regelabweichung der vorhergehenden Regelperiode verwendet.

Die Übertragungsfunktion ändert sich dabei folgendermaßen:

$$\boxed{y(t) = d \cdot (e(t_n) - e(t_{n-1}))}$$

3.2 Echtzeit

Der Begriff Echtzeit (engl.: realtime) ist ein wichtiger Aspekt einer Computersteuerung oder Regelung. Wird in der Informatik von Echtzeit gesprochen, so werden Zeitbegriffe (wie Anzahl der Takte oder Anzahl der Perioden des Algorithmus) in eine real verstrichene Zeit überführt.

Für die Regelprozesse in dieser Arbeit muss die Echtzeitbedingung gelten. Es müssen also alle Reaktionen des Experiment-PC in Echtzeit durchgeführt werden.

Definition von Echtzeit nach DIN 44300:

„Echtzeitbetrieb ist ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig derart betriebsbereit sind, dass die Verarbeitungsergebnisse innerhalb einer **vorgegebenen Zeitspanne** verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.“ [ADwin]

Soll ein Computersystem beispielsweise ein Programm in Echtzeit bearbeiten, so muss die maximal verstrichene Zeit des Programms bis zu seinem Ergebnis, also die Reaktionszeit, unterhalb einer maximal vorgegebenen Zeitspanne (meist Deadline genannt) liegen.
Der Begriff Echtzeit definiert nicht, wie viel „reale“ Zeit diese maximale Zeitspanne beträgt. Es wird nur vorausgesetzt, dass sie existiert. Die Reaktionszeit eines Computersystems kann variieren, muss aber bei Echtzeit immer unterhalb der maximal definierten Zeitspanne liegen.

Insbesondere muss ein extrem schneller Computer, nach den modernsten Technologien mit einem herkömmlichen Betriebssystem wie Windows oder Linux, nicht unbedingt echtzeitfähig sein. Zwar kann das Computersystem sehr schnell sein und im Mittel liegen seine Reaktionszeiten immer unterhalb einer Zeitspanne. Aber es ist nicht garantiert, dass diese Zeitspanne eingehalten wird.

3.2.1 Arten der Echtzeit

In Abhängigkeit der Folgen für Überschreitung der Reaktionszeit über die Regelperiodendauer unterscheidet man „harte“ und „weiche“ Echtzeitanforderungen.

Harte Echtzeit

Harte Echtzeit liegt vor, wenn ein Computersystem unter allen Umständen mit seiner Reaktionszeit unterhalb der maximalen Zeitspanne (typischerweise die Regelperiodendauer) bleibt. Mit allen denkbaren Umständen werden auch Fehlerzustände und Zustände außergewöhnlicher Betriebsbedingungen gemeint.
Sollte die maximale Zeitspanne überschritten werden so ist mit einem maximalen Schaden zu rechnen.

Diese Anforderungen scheinen übertrieben sind es aber nicht, wenn man bedenkt, dass das betroffene Computersystem einen Herzschrittmacher steuern könnte. Ein weiteres Beispiel einer Echtzeitanwendung ist beispielsweise ein Roboterarmsystem. Angenommen dieser Roboterarm soll einen extrem zerbrechlichen Gegenstand berühren, so müsste eine harte Echtzeitregelung vorgenommen werden. Denn sollte der Roboterarm durch Überschreitung der Regelperiodendauer ein Bewegung zu lange ausführen, so könnte der Gegenstand zerstört werden.

Weiche Echtzeit

Weiche Echtzeit liegt vor, wenn das Mittel aller auftretenden Reaktionszeiten, des Computersystems, unterhalb der Deadline liegt. Es existieren also vereinzelte Reaktionszeiten, welche die Deadline überschreiten. Diese Überschreitungen sind bei weichen Echtzeitanforderungen akzeptabel, da sich der auftretende Schaden in Grenzen hält.

Ein Beispiel einer solchen weichen Echtzeit ist ein Multimediasystem, zum Beispiel Video-Live-Streaming. Video-Live-Streaming liegt vor, wenn sich ein Benutzer ein Video über das Internet anschaut ohne das Video auf seine Festplatte herunter zu laden. Da ein Video eine bestimmte Anzahl von Bildern pro Sekunde benötigt, um für den Menschen flüssig dargestellt werden zu können, muss eine vorgegebenen Zeitspanne existieren, in der ein Video-Frame über das Netzwerk auf den

Computer übermittelt wird. Um diese vorgegebene Zeitspanne einzuhalten werden für solche Anwendungen spezielle Netzwerkprotokolle verwendet. Da ein Verlust eines einzelnen Video-Frames kaum vom Benutzer bemerkt wird, wenn die vorgegebene Zeitspanne nicht eingehalten wird, entsteht somit kein maximaler Schaden. Es liegt also eine weiche Echtzeitanforderung vor.

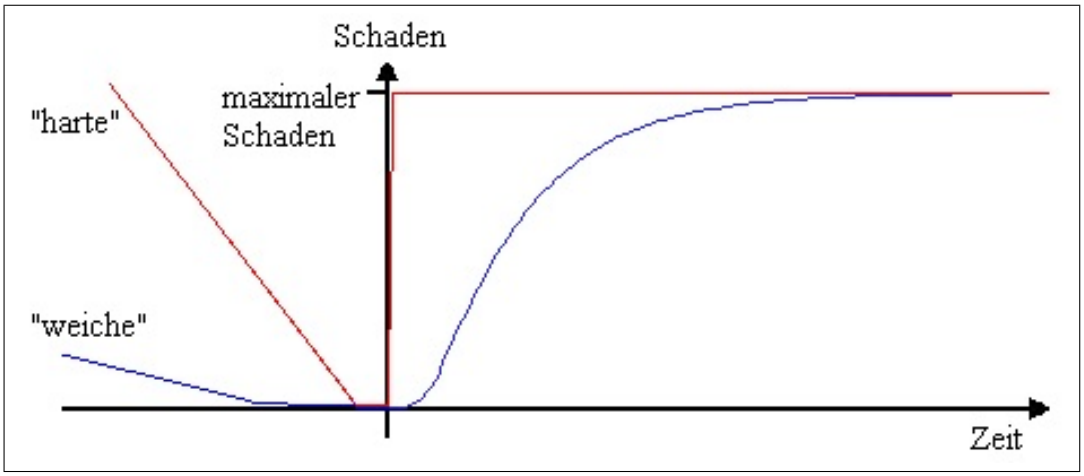

Abbildung 3.9: Schadensdiagramm vom harter (rot) und weicher (blau) Echtzeit

Abbildung 3.9 zeigt das Maß des auftretenden Schadens bei Überschreitung der Deadline. Es ist auch zu erkennen, dass eine verfrühte Regelung unerwünscht sein kann und ebenfalls zu einem Schaden führen kann. Sollte beispielsweise ein Herzschrittmacher eine verfrühte Stimulation vornehmen, so kann ebenfalls ein maximaler Schaden entstehen. Die Schadensentwicklung in Abbildung 3.9, bei verfrühter Regelung (also negativen Zeiten), ist von der konkreten Anwendung abhängig.

Das Akzeptieren von weicher Echtzeit kann die technischen Gegebenheiten eines Computersystems erheblich kostengünstiger gestalten.

3.2.2 Regelung in Echtzeit

Eine Regelung wird durch den Experiment-PC mit einem Programm durchgeführt. Das Regelprogramm realisiert beispielsweise eine PID-Regelung. Mit jeder Regelperiode berechnet der Regelalgorithmus, der PID-Regelung, einen neuen Stellwert.
Damit im Experiment eine bestimmte Regelfrequenz eingehalten werden kann, muss also der Regelalgorithmus in Echtzeit abgearbeitet werden. Dadurch wird die Zeitspanne, bis ein neuer Regelwert bereit steht, unter einer maximalen Zeitspanne gehalten.
Der Experiment-PC führt als Grundlage das Betriebsystem Linux aus. Da dieses Betriebsystem „Multitasking“ fähig ist, werden neben dem Regelalgorithmus auch andere Prozesse (Tasks) ausgeführt.
Soll nun das Regelprogramm mit dem Regelalgorithmus zur Berechnung eines neuen Stellwertes

der CPU zugeteilt werden, so benötigt der Experiment-PC eine gewisse Zeit, bis die Register und der Programmzähler der CPU in den Ausgangszustand versetzt sind. Diese Zeit wird Latenzzeit genannt. Ist die Latenzzeit verstrichen folgt die eigentliche Berechnung eines neuen Stellwertes durch den Regelalgorithmus. Die dazu benötigte Zeit wird Berechnungszeit genannt.

Die Summe der beiden Zeiten (Latenzzeit + Berechnungszeit) ergeben die Reaktionszeit. Die Reaktionszeit ist ausschlaggebend für die maximale Regelfrequenz für das Experiment.
Die folgende Abbildung 3.10 zeigt drei Verteilungen der, für die Echtzeitbedingung, relevanten Zeiten.

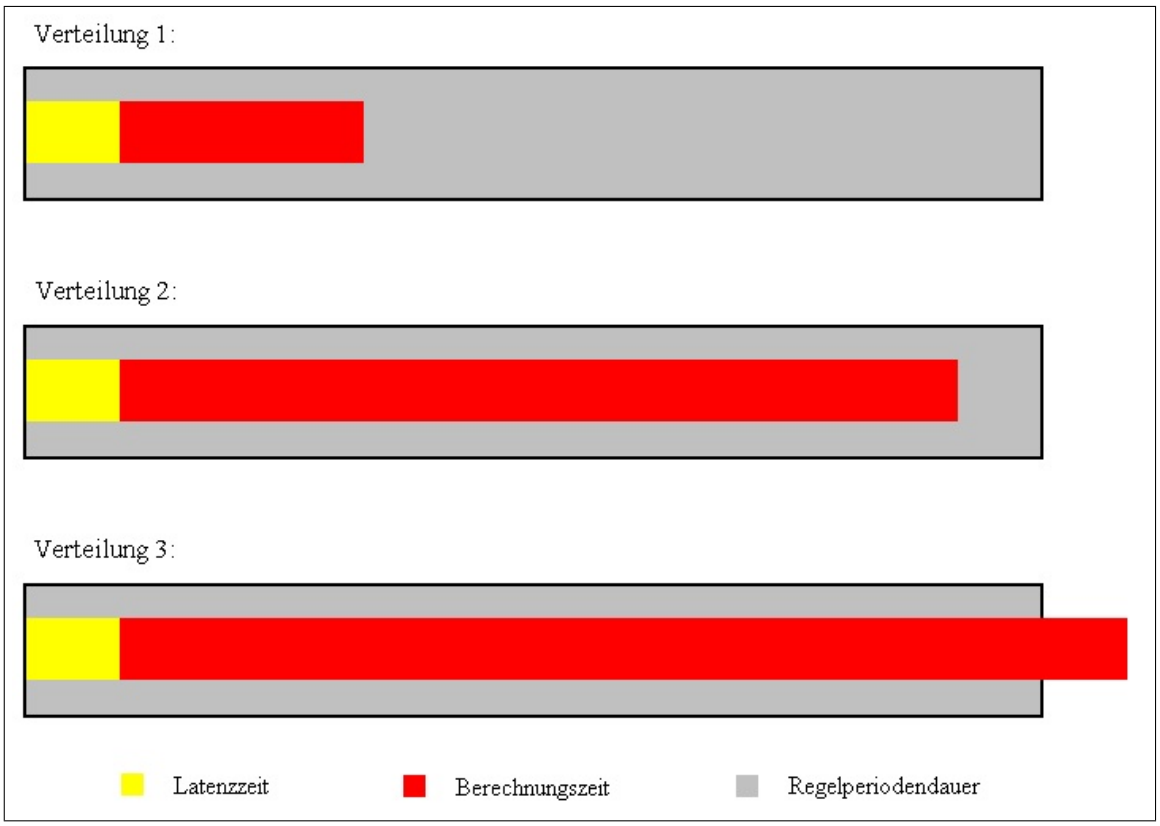

Abbildung 3.10: mögliche Verteilungen der Reaktionszeit gegenüber der Regelperiodendauer

In Abbildung 3.10 werden die maximalen Reaktionszeiten als Summe der Latenzzeiten (gelb) und der Berechnungszeiten (rot) angegeben.

Verteilung 1 zeigt ein gutes Verhältnis der Reaktionszeit zur Regelperiodendauer. Der übrige „graue“ Bereich kann durch das Linux-Betriebssystem zur Ausführung der übrigen Prozesse genutzt werden.

Verteilung 2 zeigt ein ausreichendes Verhältnis der Reaktionszeit zur Regelperiodendauer. Das Regelprogramm kann von einem Computersystem, welches die Verteilung 2 zeigt, ausgeführt werden. Allerdings bleibt dem Linux-Betriebssystem nicht genügend übrige „graue“ Zeit, in der andere Prozesse ausgeführt werden können. Das Computersystem führt die Regelung zuverlässig durch ist aber nicht mehr ansprechbar, da die freie Zeit zur Benutzerkommunikation nicht ausreicht.

Verteilung 3 zeigt ein ungenügendes Verhältnis der Reaktionszeit zur Regelperiodendauer. Die maximal auftretende Reaktionszeit überschreitet die Regelperiodendauer. Damit ist das Computersystem nicht geeignet, um eine Regelung mit dieser Regelfrequenz durchzuführen. Es muss die Regelfrequenz verringert (Regelperiodendauer erhöht) oder ein schnelleres Computersystem angeschafft werden.

3.2.3 Bedeutung für den Experiment-PC

Da mit dem Experiment-PC in dieser Arbeit harte Echtzeit umgesetzt werden soll, muss also die Reaktionszeit permanent unter der benötigten Regelperiodendauer liegen. Ob der Experiment-PC für eine Regelung in Frage kommt hängt massiv von der Geschwindigkeit des Experiment-PC ab. Dazu muss nicht nur die Geschwindigkeit der CPU ausreichen, sondern auch die des Mainboard-Chipsatzes, des RAMs, aller Caches und des Betriebsystems.

In dieser Arbeit wird lediglich der Experiment-PC für das Experiment vorbereitet. Dadurch konnte nicht getestet werden, welche Regelperiodenzeit konkret für das Experiment benötigt wird.
Im Kapitel 7 werden die wichtigen Latenzzeiten für den, in dieser Arbeit, verwendeten Experiment-PC angegeben.

3.2.3.1 Echtzeit unter Linux

Die Erläuterungen in diesem Kapitelpunkt sind dem Kapitel 4 vorweggenommen. Diese Erläuterungen sind aber dem besseren Verständnis dienlich.
Nach der vollständigen Vorbereitung des Experiment-PC ergeben sich zwei Möglichkeiten Programme, zu denen auch ein Regelprogramm gehört, in Echtzeit auszuführen. Grundsätzlich können Programme unter Linux im **User-Space** oder im **Kernel-Space** ausgeführt werden. Die Begriffe User-Space und Kernel-Space sind eher Bezeichnungen für Modi in denen sich ein Programm befinden kann, aus diesem Grund werden die Begriffe mit User-Mode oder Kernel-Mode gleichgesetzt.
Der User-Space ist die oberste Ebene der Abstrahierung und der Kernel-Space die unterste Ebene. Programme in User-Space sind meist gewöhnliche Anwendungsprogramme und werden beispielsweise mit „`./[programm]`“[1] unter Linux gestartet. Der Kernel-Space befindet sich am nächsten zur Hardware.
Programme im Kernel-Space werden auch Kernelmodule genannt und müssen unter Linux mit „`$ insmod [programm-modul]`“[2] oder „`$ modprobe [programm-modul]`“ geladen werden.

[1] Der Ausdruck `[programm]` steht für die ausführbare Datei eines Programms.
[2] Der Ausdruck `[programm-modul]` steht bei „insmod“ für den vollständigen Namen eines Kernelmoduls (Bsp.: „kcomedilib.ko“) und bei „modprobe“ für den Vornamen des Kernel-Moduls (Bsp.: „kcomedilib“).

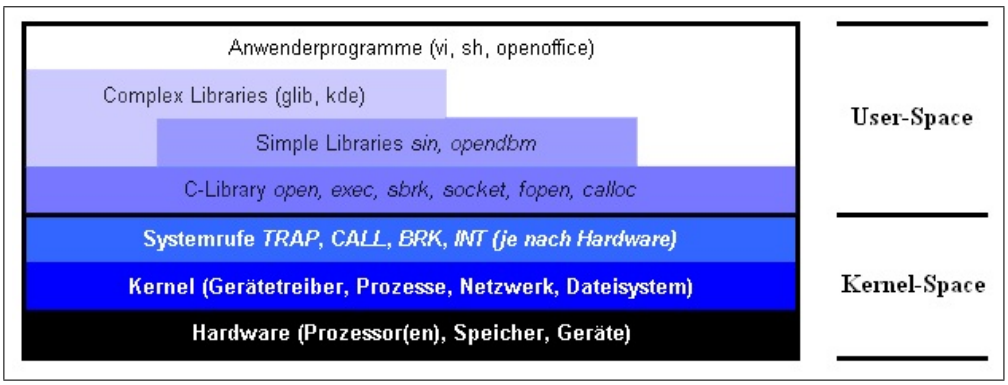

Abbildung 3.11: Kernel- und User-Space von Linux

Abbildung 3.11 zeigt den prinzipiellen Schichtenaufbau eines Linux-Betriebssystems. Nur Kernel-Module können auf die Hardware zugreifen. Programme im User-Space müssen mit Kernel-Modulen kommunizieren, um auf die Hardware zuzugreifen.

User-Space

Werden Programme im User-Space ausgeführt, so erhalten diese Programme eine niedrigere Privilegierung und werden in einem Speicherbereich gehalten, welcher sich nicht mit dem Kernel-Space überlappt.
Typische Programme, welche im User-Space ausgeführt werden, sind Anwendungsprogramme wie Schreibprogramme oder Spiele. Zum größten Teil sind dies Programme, welche über ein **G**raphical **U**ser **I**nterface (kurz GUI) verfügen. Programme in User-Space können nicht direkt auf die Hardware des Computers zugreifen, sondern müssen Hardwarekommunikation mittels eines **A**pplication **P**rogramming **I**nterface (kurz API) durchführen. Dieses API nutzt sogenannte „Gates“, welche „Löcher“ im Speicher darstellen, um mit Programmen im Kernel-Space zu kommunizieren.

Kernel-Space

Programme im Kernel-Space können direkt auf die Hardware des Computers zugreifen und erhalten dadurch eine schnellere Kommunikationsmöglichkeit mit den Peripheriegeräten des Computers. Aus diesem Grund sind Gerätetreiber Programme, welche im Kernel-Space laufen.
Programme in Kernel-Space besitzen meist höhere Privilegien und Prioritäten als Programme im User-Space.
Die Aufspaltung von Programmen nach User- und Kernel-Space hilft dabei die Hardware zu abstrahieren und bietet bessere Möglichkeiten, um Anwendungsprogramme und Treiber unabhängig

voneinander zu programmieren. Zusätzlich werden die Speicherbereiche einzelner Prozesse voneinander abgeschottet.

Echtzeitprogramme können nun im Kernel-Space oder User-Space ausgeführt werden. Dabei wird die Variante des Kernel-Space bevorzugt. Vor allem, wenn ein Programm mit einer Hardware kommuniziert, wie im Beispiel dieser Arbeit, so ist es ratsam diese Programme im Kernel-Space auszuführen.
Wird nun ein Programm in Linux gestartet, so wird dem Programm ein einzelner Prozess zugeordnet. Dieser Prozess kann dann unabhängig von anderen Prozessen ausgeführt werden. Erhält nun ein Prozess Rechenzeit auf der CPU so muss ein Kontextwechsel durchgeführt werden. Dieser Kontextwechsel verursacht unter anderem auch die, in Kapitelpunkt 3.2.2 erwähnte, Latenzzeit. Läuft nun ein Echtzeit-Programm im User-Space und will mit einer Hardware kommunizieren, so muss ein Kontextwechsel zu einem Programm im Kernel-Space eingeleitet werden, da nur Programme im Kernel-Space auf die Hardware zugreifen können.
Dieser zusätzliche Kontextwechsel, welcher bei einem Echtzeit-Programm im Kernel-Space nicht nötig wäre, erhöht die Latenzzeit. Um die zusätzliche Latenzzeit zu vermeiden werden Echtzeitprogramme meist als Kernelmodul gestartet.

Um Echtzeitprogramme im User-Space auszuführen kann das Softwareelement LXRT genutzt werden.

3.2.4 Bedeutung von Interrupts

Interrupts sind Unterbrechersignale, welche von einem Interrupt-Controller verarbeitet werden. Der Interrupt-Controller ist eine logische Schaltung, also Hardware, und empfängt alle ausgelösten Interrupts eines Systems. Der Interrupt-Controller verarbeitet die Interrupts und leitet sie an die CPU des Systems weiter. Die CPU kann dann eine entsprechende Interruptbehandlungsroutine ausführen, um entsprechend auf den Interrupt zu reagieren.

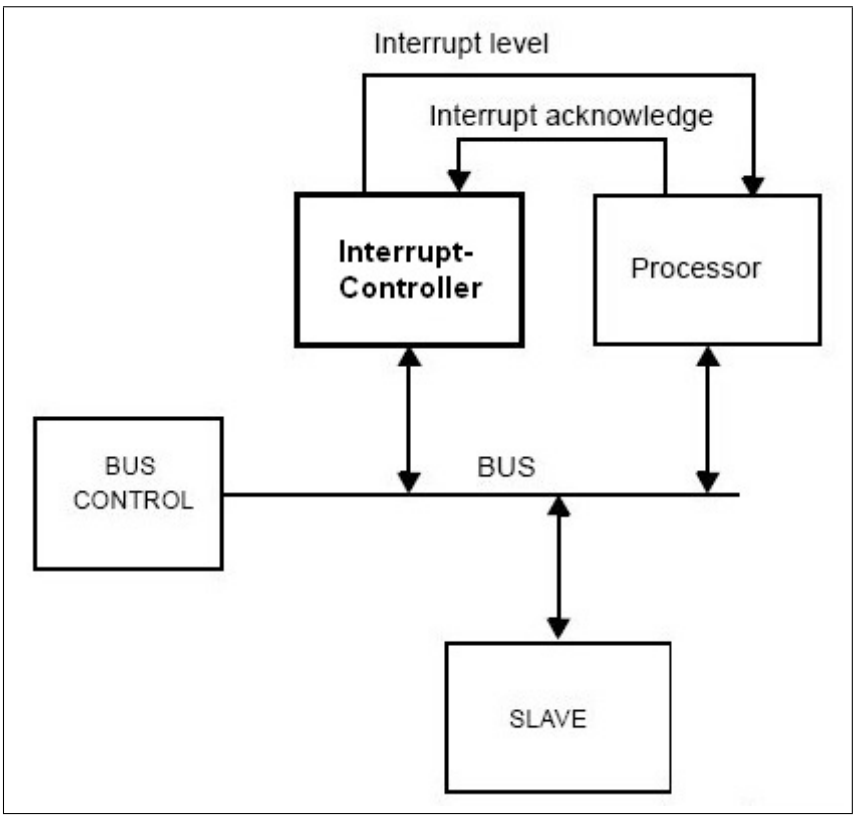

Abbildung 3.12: Interrupt-Controller in einem System

In Abbildung 3.12 ist zu erkennen, dass bei einem auftretenden Interrupt ein „Interrupt Level“ durch den Interrupt-Controller erzeugt wird, der dann zur CPU getrieben wird. Das Interrupt Acknowledgement (deutsch.: Interrupt Bestätigung) wird von der CPU in den Interrupt-Controller getrieben, so dass festgestellt werden kann wann die CPU die Behandlung des Interrupts übernimmt. Ein Interrupt kann durch die Slaves am System-Bus über Hardwareleitungen oder per Software, über Register, ausgelöst werden.

Interrupts haben eine essentielle Bedeutung für effiziente Computersysteme oder Echtzeit-Systeme. Angenommen es wird von einem Computersystem ein Programm ausgeführt, welches Alarm auslösen soll, sobald die Temperatur der CPU über einen bestimmten Wert steigt. Dies setzt natürlich voraus, dass es einen Sensor gibt, der die Temperatur der CPU ermitteln kann.
Das Programm, was den Alarm auslösen soll, läuft im User-Space und hat die Aufgabe spätestens 0,1 Sekunden nachdem die kritische Temperatur überschritten wurde den Alarm auszulösen. Davon abgesehen, dass die Einhaltung der Aufgabe ohne Echtzeit nicht zu garantieren ist, gilt das gegebene Beispiel auch für nicht-Echtzeit-Programme.

Ohne Interrupt-Controller und damit ohne Interrupts müsste das Alarmprogramm theoretisch aller 0,1 Sekunden überprüfen, ob die kritische Temperatur der CPU überschritten wird. Dabei sei die Berechnungszeit des Programms vernachlässigt. Das ständige Überprüfen der Temperatur wird „Polling“ genannt und belastet die CPU unnötig, da es für lange Zeit vorkommen kann, dass die kritische Temperatur nicht überschritten wird. In all dieser Zeit erhält das Alarmprogramm

CPU-Zeit, obwohl es keinen Alarm auslösen muss.

Mit Interrupt-Controller kann von einer speziellen Hardware, welche den Temperatursensor der CPU überwacht, ein Interrupt ausgelöst werden. Dieser Interrupt wird meist mit einer konkreten Zahl assoziiert, wie zum Beispiel „IRQ 5“. Erst wenn der Interrupt 5 ausgelöst würde, leitet der Interrupt-Controller ein Signal an die CPU des Systems weiter. Die CPU kann dann das Alarmprogramm ausführen und dieses kann dann den gewünschten Alarm ausführen.
Es wird also CPU-Zeit gespart und das Alarmprogramm nur ausgeführt, wenn die kritische Temperatur überschritten wurde.
Interrupts können durch Hardware als auch durch Software ausgelöst werden. Bei der Auslösung durch Hardware werden Verdrahtungen genutzt, die speziell für Interrupts reserviert werden. Bei der Auslösung durch Software werden Daten in die Register des Interrupt-Controllers geschrieben. Hardware- und Software-Interrupts werden meist gleichberechtigt behandelt.

Interrupts sind ebenfalls bei Echtzeit-Programmen nützlich. Da Echtzeit-Tasks empfindliche Programme sind ist es wichtig sie nur auszuführen, wenn sie benötigt werden. Viele Regelprogramme müssen zum Beispiel nach einer bestimmten Regelperiodendauer wieder ausgeführt werden. Damit das Regelprogramm nicht ständig, über Polling, die Systemzeit auslesen muss, um festzustellen wann die Regelperiode abgelaufen ist, kann ein Interrupt verwendet werden, um das Regelprogramm zu reaktivieren. Der Interrupt wird ausgelöst, sobald die Regelperiodendauer verstrichen ist.

Kapitel 4

Umsetzungsmöglichkeiten der Regelung in Echtzeit

Der Leser kann sich vorstellen, dass eine Regelung mit normalen passiven oder aktiven elektrischen Bauteilen, aufgrund der Einfachheit der verwendeten Bauteile, ein hoher Zuverlässigkeitsgrad zur Erfüllung der Echtzeitanforderung besitzt.
Wie kann aber diese Echtzeitanforderung von einem Computersystem wie dem Experiment-PC erfüllt werden? Umsetzungsmöglichkeiten, die dies ermöglichen werden in diesem Kapitel besprochen.

4.1 Comedi

Das „Comedi“ Projekt ist ein Open-Source-Projekt und ist speziell für Linux erhältlich. Die Entwickler von Comedi haben es sich zum Ziel gemacht ein einheitliches API zu erstellen, welches Mess- und Steuerkarten vieler verschiedener Firmen ansteuern kann.
Da viele Firmen von Mess- und Steuerkarten nur die Treiber ihrer eigenen Karten veröffentlichen, ist es schwer ein Regel- oder Steuerprogramm unter Linux zu programmieren, welches auf verschiedenen PCs mit unterschiedlichen Mess- und Steuerkarten erfolgreich ausführbar ist.
Ein einheitliches Interface, wie das von Comedi, erleichtert die Arbeit des Programmierers erheblich.
In dieser Arbeit wurden die Softwareelemente „Comedi“ in der Version 0.7.74 und „Comedilib“ in der Version 0.8.0 verwendet.

Das Comedi Projekt unterteilt sich in drei Softwareelemente, welche für die Echtzeit-Regelung relevant sind:

1. **Comedi** ist eine Sammlung von Treibern für verschiedene Mess- und Steuerkarten. Der Comedi Treibersammlung ist also bekannt, welche Funktionalitäten von einer bestimmten Mess- und Steuerkarte unterstützt werden. Zu diesen Funktionalitäten gehören zum Beispiel die Anzahl der verfügbaren analogen Ausgänge oder Eingänge.
   Damit das Comedi-Interface für eine bestimmte Mess- und Steuerkarte verwendet werden kann, muss der Treiber der Karte in der verwendeten Comedi-Version verfügbar sein.

   Die aktuell unterstützen Mess- und Steuerkarten sind auf der Homepage: http://www.comedi.org/hardware.html aufgelistet.

2. **Comedilib** auch „Comedi Library“ genannt ist eine Entwickler-Bibliothek, mit der eine API zum Zugriff auf die Mess- und Steuerkarte bereitgestellt wird. Ebenfalls können Programmierer mit ihr einfache Konfigurationen und Kalibrierungen, der einzelnen Comedi-Devices, durchführen. Zusätzlich enthält die Comedilib Demonstrationsprogramme, die alle grundlegenden Funktionen beleuchten.

3. **kcomedilib** ist ein Kernelmodul, dass ein spezielles Interface wie Comedilib zur Verfügung stellt und von Kernel-Modulen genutzt werden kann. Dadurch wird es möglich ein Kernel-Modul zu programmieren, welches auf die Mess- und Steuerkarte zugreifen kann.

4.1.1 Handhabung und Funktionsweise

Comedi ordnet jeder Mess- und Steuerkarte, welche sich in einem PC befindet, ein sogenanntes **Device** zu. Die Devices werden von 0 an beginnend indiziert (comedi0, comedi1 ... comedi(n)) und im Linux-Betriebssystem im Verzeichnis „`/dev/`“ als „Dateien“ abgelegt. Nun kann das Betriebssystem durch gewöhnliche Dateiverwaltungen und Verarbeitungen mit der jeweiligen Mess- und Steuerkarte kommunizieren.
Comedi teilt jedem Device eine verschiedene Anzahl von **Subdevices** zu. Diese Subdevices werden ebenfalls von 0 an beginnend indiziert und stellen eine Klassifikation aller Funktionalitäten der Mess- und Steuerkarte dar. Die Klassifikation wird nach analogen Eingängen, analogen Ausgängen, digitale Ein-/Ausgängen (engl.: **I**n-/**O**utputs kurz I/O), sowie Counter und Timer durchgeführt.
Besitzt eine Mess- und Steuerkarte nun 16 analoge Eingänge, so sind diese beispielsweise alle unter einem Subdevice0 zusammengefasst.
Alle Bestandteile eines Subdevices, beispielsweise die 16 analogen Eingange, werden als Channels bezeichnet. Ein Subdevice0, mit 16 analogen Eingängen, besitzt also 16 Channels.

Abbildung 4.1 zeigt die Ausgabe des Comedi-Programms „`board_info`“ im „`/usr/local/src/comedilib/demo/`“ Verzeichnis des Experiment-PC:

```
overall info:

    version code: 0x00074a
    driver name: me4000
    board name: ME-4660i
    number of subdevices: 4

subdevice 0:

    type: 1 (analog input)
    flags: 0x00719000
    number of channels: 16
    max data value: 65535
    ranges:
    all chans: [0,2.5] [0,10] [-2.5,2.5]
    [-10,10]
    command:
    start: now|ext
    scan_begin: follow|timer|ext
    convert: timer|ext
    scan_end: none|count
    stop: none|count

subdevice 1:

    type: 2 (analog output)
    flags: 0x00320000
    number of channels: 2
    max data value: 65535
    ranges:
    all chans: [-10,10]

subdevice 2:

    type: 5 (digital I/O)
    flags: 0x00030000
    number of channels: 32
    max data value: 1
    ranges:
    all chans: [0,5]

subdevice 3:

    type: 6 (counter)
    flags: 0x00030000
    number of channels: 3
    max data value: 65535
    ranges:
    all chans: [0,1]
```

Abbildung 4.1: Subdevices und Channels der ME-4660i durch „`board_info`“

In Abbildung 4.1 werden alle Subdevices und Channels der Mess- und Steuerkarte (ME-4660i) des Experiment-PC angezeigt.

Comedi stellt verschiedenste C++ Befehle der Comedi-API zur Verfügung, um Daten mit der Mess- und Steuerkarte auszutauschen. Die Befehle der Comedi-API unterteilen sich in Befehle für den User-Space und Kernel-Space.

User-Space	Kernel-Space	Bedeutung
`comedi_data_read()`	`rt_comedi_datain()`	Zum Einlesen eines elektrischen Signals
`comedi_data_write()`	`rt_comedi_dataout()`	Zum Ausgeben eines elektrischen Signals

Tabelle 4.1: äquivalente RTAI-Befehle aus User-Space und Kernel-Space

In Tabelle 4.1 ist zu erkennen, dass die Befehle für den Kernel-Space mit „rt_“ beginnen und dadurch für den Echtzeitbetrieb gekennzeichnet sind. Befehle für den User-Space nutzen die API der Comedilib und kommunizieren mit der Mess- und Steuerkarte über die Linux-Devices und das Kernelmodul „comedi“. Die Befehle für den Kernel-Space nutzen zur Kommunikation mit der Mess- und Steuerkarte das Kernelmodul „kcomedilib“.

4.1.2 Comedi in Echtzeit

Wie bereits erwähnt sind die Echtzeitbefehle von Comedi hauptsächlich für den Kernel-Space gedacht. Nur diese Echtzeitbefehle, welche mit „rt_“ beginnen, können von dem Kernel-Modul „kcomedilib“ in Echtzeit verarbeitet werden. Um Comedi für Kernel-Module nutzen zu können müssen die Kernel-Module „comedi“, „kcomedilib“ und „comedi_fc“ mit „insmod“ oder „modprobe“ geladen werden.

Die Befehle, welche nicht mit „rt_“ beginnen, sind nur für einfache Anwendungen gedacht, die nicht in Echtzeit verarbeitet werden sollen. Da die normalen User-Space-Befehle von Comedi allerdings hauptsächlich auf der Mess- und Steuerkarte operieren, welche selbst nur durch Hardware realisiert wird, ergibt sich eine relativ zuverlässige und geringe Latenzzeit dieser Befehle. Dies darf allerdings nicht mit konkreter Echtzeit verwechselt werden.

4.2 LabVIEW

Die Software LabVIEW der Firma National Instruments ist eine bekannte und weit verbreitete Mess- und Automatisierungssoftware. Mit ihr können diverse Steuerungen oder Regelungen über ein benutzerfreundliches GUI erstellt werden. Alle Steuerungen können in sogenannten virtuellen

Instrumenten abgelegt werden. Diese virtuellen Instrumente können über graphische Elemente, wie Regler und Anzeigen, gesteuert werden.

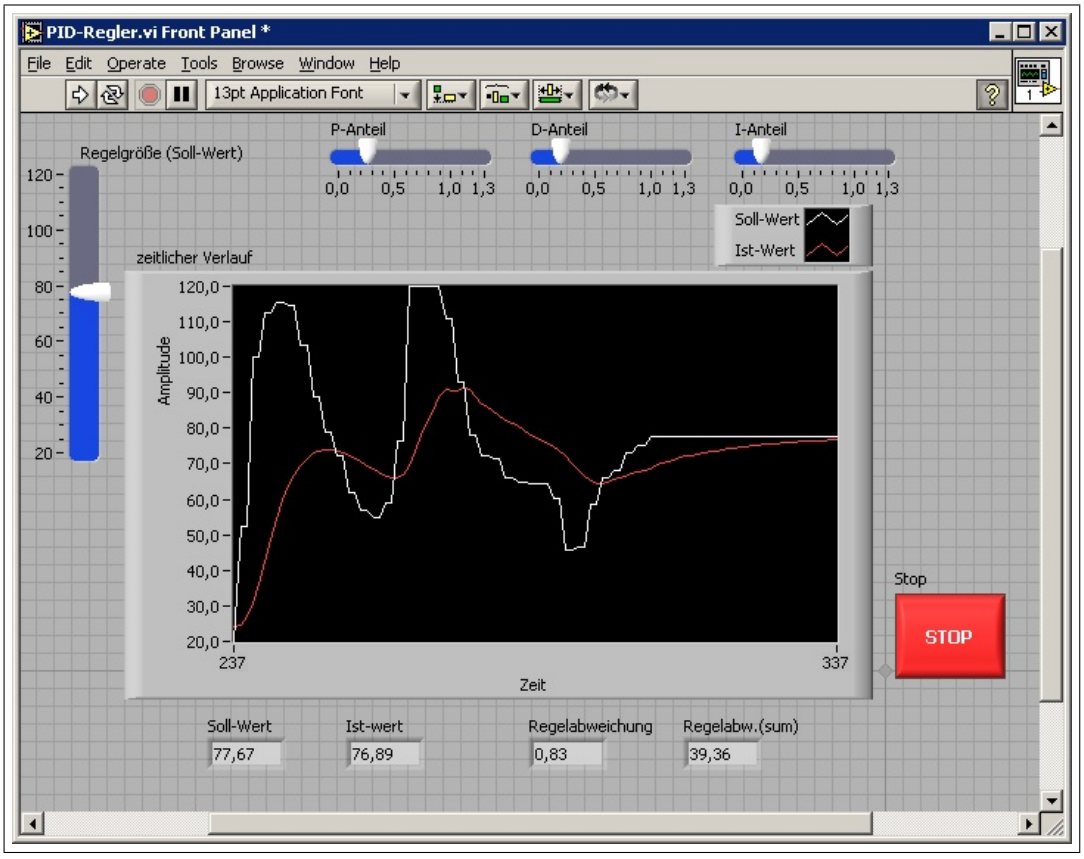

Abbildung 4.2: Steuer-GUI einer PID-Regelung in LabVIEW

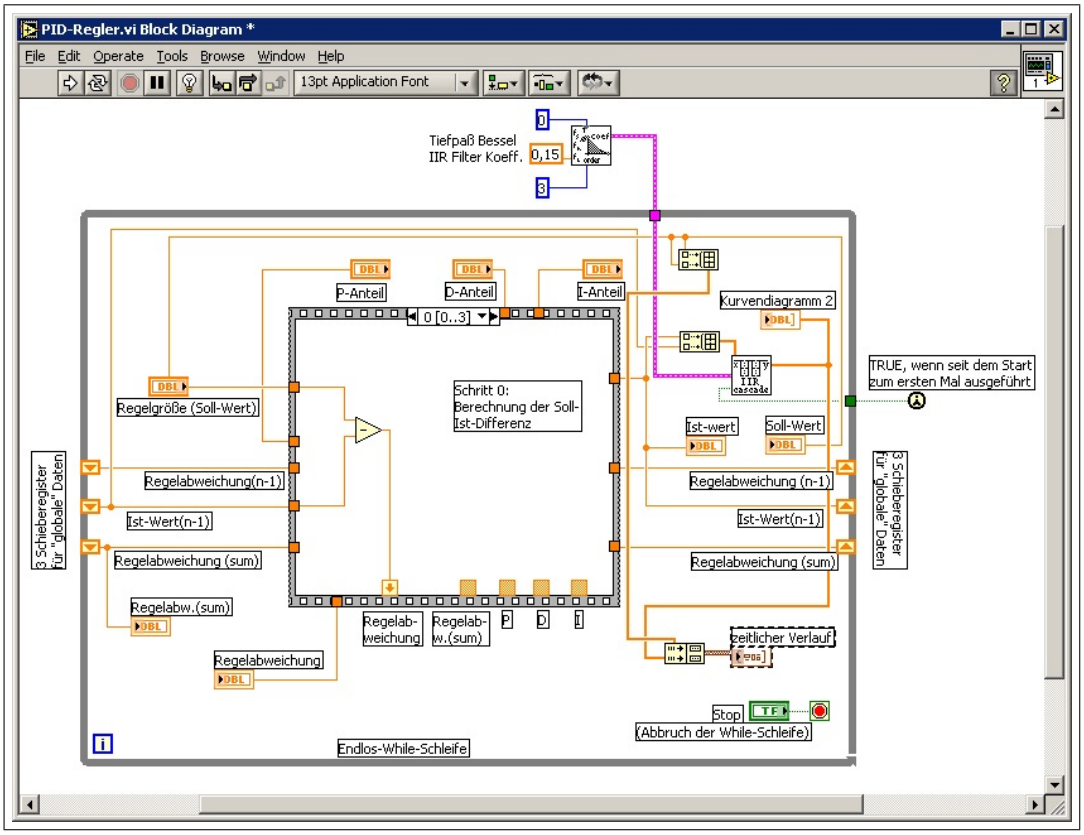

Abbildung 4.3: Datenfluss-GUI einer PID-Regelung in LabVIEW

Abbildung 4.2 und 4.3 zeigen die grafischen Hauptelemente von LabVIEW, die Steuer-GUI und die Datenfluss-GUI. Diese GUIs wurden aus dem virtuellen Instrument einer PID-Regelung erstellt.

Alle Steuerungen von LabVIEW können über eine Mess- und Steuerkarte ausgegeben werden. Leider werden standardmäßig nur Mess- und Steuerkarten der Firma National Instruments unterstützt. Soll eine firmenfremde Steuerkarte benutzt werden, so muss diese mit zusätzlichem Aufwand installiert werden.
In dieser Arbeit ist es nicht gelungen die Software LabVIEW in Verbindung mit der Meilhaus Mess- und Steuerkarte (ME-4660i) unter Echtzeitanforderungen zu benutzen. Nähere Erläuterungen zu diesem Ergebnis können in Kapitelpunkt 7.2 nachgelesen werden.

Echtzeit-Regelung unter LabVIEW

Soll eine Echtzeit-Regelung mit LabVIEW realisiert werden, so wird im Folgenden vorrausgesetzt, dass eine Mess- und Steuerkarte der Firma National Instruments im Computersystem vorhanden ist. Es werden im Folgenden weiter Angaben über die Software LabVIEW gemacht, welche im Rahmen dieser Arbeit durch den speziellen Kundendienst der Firma National Instruments erfragt wurden. Es wird darauf hingewiesen, dass diese Angaben nicht genau oder gar falsch sein können. Der folgende Abschnitt ist somit keine offizielle Quelle über Produkte der Firma National Instruments.

In dieser Arbeit lag das LabVIEW in der Version 7.1 für das Betriebssystem Linux vor. Allein mit dieser Version konnte keine Echtzeit realisiert werden. Für Echtzeit-Regelungen müsste ein zusätzliches Real-Time-Modul der Firma National Instruments gekauft werden, welches die Möglichkeit bietet Echtzeit zu nutzen. Dies wurde in dieser Arbeit nicht getan.
Die Firma National Instruments legt besonderen Wert auf hohe Zuverlässigkeit ihrer Echtzeit-Systeme und hat, aus Stabilitätsgründen, davon Abstand genommen, Echtzeit-Programme und deren Erstellung in LabVIEW auf einem einzigen Computersystem zu betreiben.
Der „neue“ Workflow von National Instruments sieht es vor auf einem speziellen Host-System, auf dem LabVIEW mit dem Real-Time-Modul installiert wurde, den Regelprozess zu entwerfen. Ist der Regelprozess fertiggestellt und gegebenenfalls kompiliert, so wird er in ein Target-System eingebracht. Das Target-System ist ein spezielles Echtzeit-Computersystem, welches einzig und allein für die Ausführung der Echtzeit-Anwendung benutzt wird.

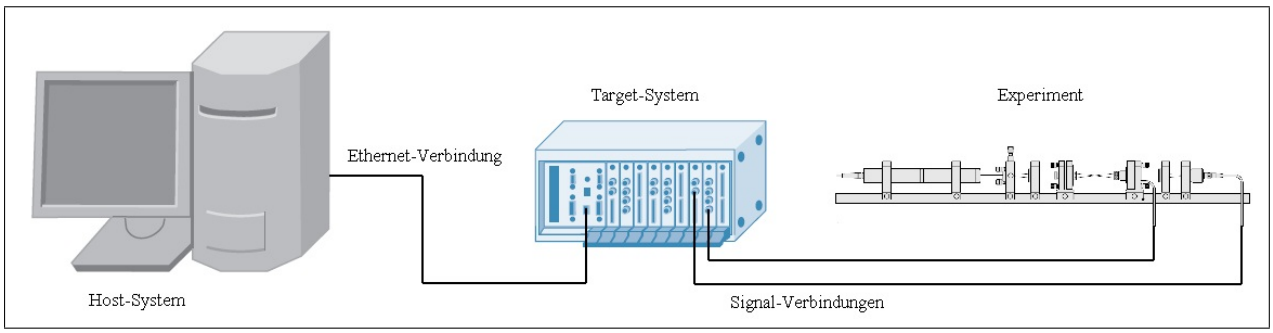

Abbildung 4.4: Host- und Target-System für LabVIEW Echtzeitumsetzungen

Es ist in der Abbildung 4.4 zu erkennen, dass das Host-System nicht an einen Versuchsaufbau angeschlossen wird. Auf dem Host-System kann Linux oder Windows installiert sein. Wurde ein Steuerungsprogramm, welches in Echtzeit ausgeführt werden soll, mit dem Hostsystem entworfen, so wird es über Netzwerk auf das Target-System übertragen.
Auf dem TargetSystem läuft ein reines Echtzeit-Betriebssystem. Zu reinen Echtzeit-Betriebssystemen zählen weder Linux noch Windows. Das Echtzeit-Betriebssystem ist, mit einer speziell abgestimmten Hardware, in der Lage den Echtzeit-Task zuverlässig in hohen Frequenzen auszuführen. Das Target-System ist dabei über eine Mess- und Steuerkarte mit dem Versuchsaufbau verbunden.

4.3 Echtzeit-Betriebssysteme

Um Echtzeit-Programme auf einem PC ausführen zu können gibt es, wie bei einem Target-System des LabVIEW-Konzeptes, die Möglichkeit ein Echtzeit-Betriebssystem auf dem PC zu benutzen. Bekannte Echtzeit-Betriebssysteme sind „eCos“ oder „VxWorks“.

4.3.1 „eCos“ Betriebssystem

eCos, die Abkürzung für „**e**mbeded **C**onfiguration **o**perating **s**ystem“, ist ein Open-Source Echtzeit-Betriebssystem, welches hauptsächlich in ES Verwendung findet.
Das Betriebssystem eCos ist ein Nicht-Linux Betriebssystem und bietet in erster Linie hohe Konfigurierbarkeit und Portabilität. Es wird eine **H**ardware **A**bstraction **L**ayer (kurz HAL) benutzt, um eine unabhängige Schnittstelle von Software zur Hardware zu erhalten. Dies erhöht die Portabilität.
eCos wird auf einem Host-PC, mit Linux oder Windows, konfiguriert und kompiliert, um dann auf das Zielsystem übertragen zu werden. Eine Richtlinie bei der Entwicklung des eCos Betriebssystems ist der geringe Speicherbedarf.

4.3.2 „VxWorks“ Betriebssystem

VxWorks ist ein weiteres Echtzeit-Betriebssystem, welches auf unterschiedlichen CPU-Plattformen eingesetzt werden kann. Das Betriebssystem VxWorks 6.3 wird auf einem externen Host-PC erstellt und kompiliert.
VxWorks findet hauptsächlich in Kleingeräten Anwendung. Solche Kleingeräte sind beispielsweise kleinere Maschinen in Luft- und Raumfahrt oder in Bereichen der Medizin.

4.4 Real Time Application Interface (RTAI)

Das **R**eal **T**ime **A**pplication **I**nterface (kurz RTAI) ist ein Open-Source-Projekt zur Erweiterung eines gewöhnlichen Linux Betriebssystems zu einem Echtzeit-Betriebssystem. Die Homepage des RTAI-Projektes ist http://www.rtai.org, dort kann die neuste Version der RTAI-Erweiterung heruntergeladen werden.
Grundsätzlich wird bei einer RTAI-Erweiterung eines gewöhnlichen Linux-Betriebsystems ein RTAI-Patch auf den Sourcecode des Linux-Kernels angewandt. Dieser Patch fügt Schnittstellen in den Linux-Kernel ein, welche die Echtzeiterweiterung ermöglichen. Der gepatchte Linux-Kernel muss dann neu kompiliert werden und beim Systemstart bebootet werden.

4.4.1 Aufbau

Die RTAI-Erweitungen fügt einen zusätzlichen Echtzeit-Kernel (auch RT-Kernel) zwischen Hardware und Linux-Kernel ein. Dieser RT-Kernel übernimmt hauptsächlich die Interrupt-Verwaltung und das Scheduling der Echtzeit-Tasks.

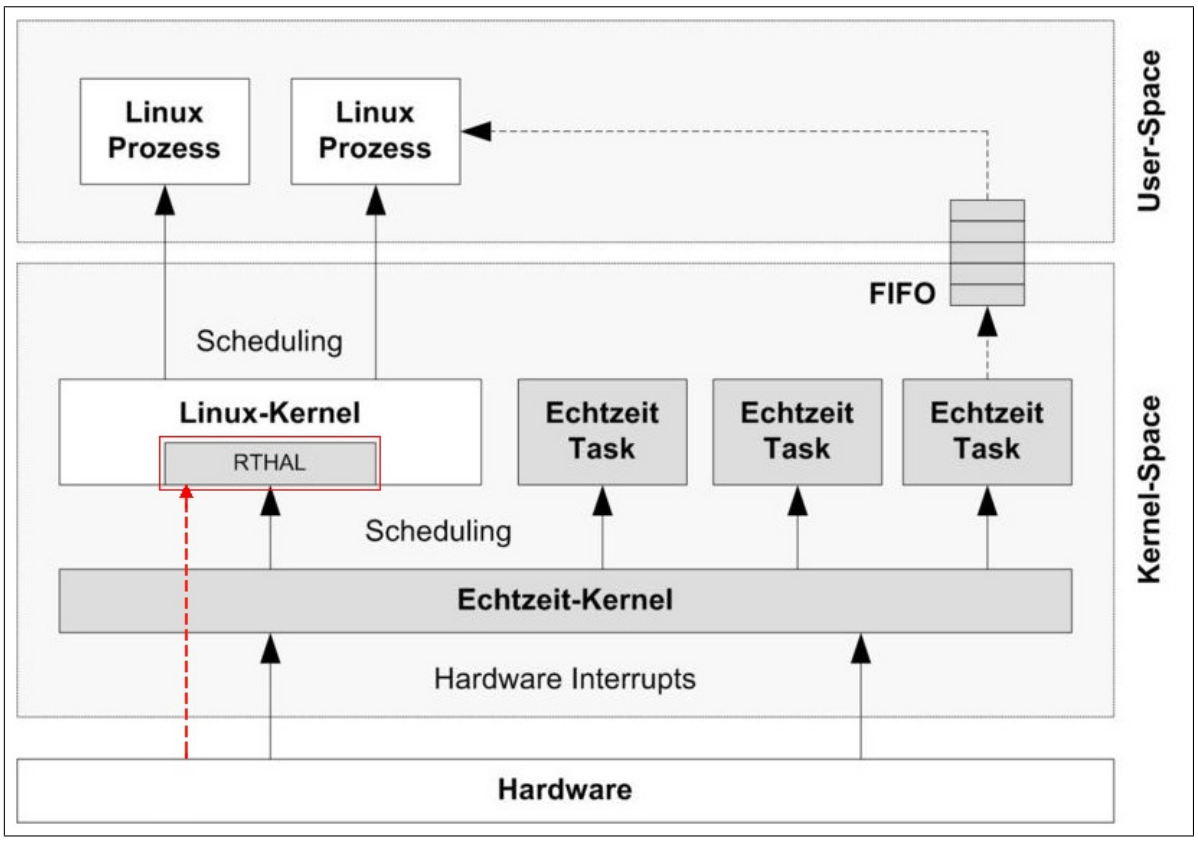

Abbildung 4.5: Aufbau eines Linux mit RTAI-Erweiterung

Abbildung 4.5 zeigt den neuen Gesamtaufbau des Softwaresystems, nachdem die RTAI-Erweiterung in das Linux-Betriebssystem eingebracht wurde. Alle grauen Elemente sind Neuerungen der RTAI-Erweiterung.

Der Echtzeit-Kernel empfängt nun zuerst alle Hardware-Interrupts des Systems, vor dem normalen Linux-Kernel und verwaltet die Echtzeit-Tasks, welche am Echtzeit-Kernel angemeldet werden müssen. Der normale Linux-Kernel wird in einem RTAI-System als ein Echtzeit-Task mit der niedrigsten Priorität behandelt und wird nur dann ausgeführt, wenn kein anderer Echtzeit-Task die CPU benötigt. Die normalen Linux-Prozesse werden weiterhin unberührt durch den Scheduler des Linux-Kernels verwaltet. Die Verwaltung geschieht allerdings nur, wenn dem Linux-Kernel selbst CPU-Zeit zugeteilt wurde.

4.4.2 Realtime Hardware Abstraction Layer (RTHAL)

Die **R**eal **T**ime **H**ardware **A**bstraction **L**ayer (kurz RTHAL) wird durch das RTAI-Erweiterungs-Patch in den Sourcecode des herkömmlichen Linux-Kernels integriert und steht dann, nach dem Booten des neu kompilierten Linux-Kernels, zur Verfügung. Die RTHAL ist in der Abbildung 4.5 rot umrahmt.
Die RTHAL ermöglicht es den RT-Kernel zwischen der Hardware und dem Linux-Kernel einzubringen. Der RT-Kernel verwaltet den Linux-Kernel über die RTHAL und sendet alle Interrupts an

diesen weiter, wenn sie nicht an die Echtzeit-Tasks gerichtet sind.
Durch die RTHAL ist es weiter möglich die Echtzeitfähigkeit des verwendeten Linux zu deaktivieren. Dazu werden alle Hardware-Interrupts und Signale nun direkt an den Linux-Kernel weitergeleitet. Dies ist mit dem roten Pfeil in der Abbildung 4.5 dargestellt.

4.4.3 RTAI-Module

Wurde die RTAI-Erweiterung vollständig auf dem Linux-System eingerichtet, so stehen spezielle RTAI-Kernel-Module (auch RTAI-Module) zur Verfügung. Sie werden standardmäßig im Verzeichnis „`/usr/realtime/modules`“ abgespeichert.

```
insmod/usr/realtime/modules/rtai_hal.ko
insmod/usr/realtime/modules/rtai_up.ko # oder rtai_lxrt.ko
insmod/usr/realtime/modules/rtai_fifos.ko
insmod/usr/realtime/modules/rtai_sem.ko
insmod/usr/realtime/modules/rtai_mbx.ko
insmod/usr/realtime/modules/rtai_msg.ko
insmod/usr/realtime/modules/rtai_netrpc.ko ThisNode="127.0.0.1"
insmod/usr/realtime/modules/rtai_shm.ko
insmod/usr/realtime/modules/rtai_leds.ko
insmod/usr/realtime/modules/rtai_signal.ko
insmod/usr/realtime/modules/rtai_tasklets.ko
insmod/usr/realtime/modules/rtai_comedi.ko
```

Abbildung 4.6: Laden der RTAI-Module laut [rtaitut]

Werden die RTAI-Module aus Abbildung 4.6 mit „insmod“ oder „modprobe“ geladen, so wird der RT-Kernel des Systems aktiviert. Alle Hardware-Interrupts und Signale werden ab jetzt zuerst vom RT-Kernel verarbeitet. Das Laden der RTAI-Module aus Abbildung 4.6 stellt die volle Funktionalität vom RTAI her.

Ausgewählte RTAI-Module

Der Name aller RTAI-Module beginnt mit „`rtai_`“. „`rtai_up.ko`“ und „`rtai_lxrt.ko`“ sind spezielle RTAI-Scheduler. Mit dem Modul „`rtai_lxrt.ko`“ ist es möglich LXRT zu nutzen und damit Echtzeit-Tasks im User-Space auszuführen. „`rtai_up.ko`“ ist ein Scheduler für Uni-Prozessor-Systeme, also für Computer mit nur einer CPU. Für Multi-Prozessor-Systeme können stattdessen die Module „`rtai_smp.ko`“ oder „`rtai_mup.ko`“ geladen werden. SMP ist die Abkürzung für Symetric Multiprocessing und MUP steht für **M**ulti-**U**ni-**P**orzessor. Das Modul „`rtai_up.ko`“ wurde in dieser Arbeit verwendet.
Module zur Kommunikation zwischen Tasks sind „`rtai_mbx.ko`“ (**M**ail**box**), „`rtai_sem.ko`“

(**Sem**aphore) und „`rtai_fifos.ko`“ (**F**irst-**I**n-**F**irst-**O**ut-Datenstruktur). Mailboxen und Semaphore werden zur Kommunikation zwischen Echtzeit-Tasks genutzt und mit FIFOs ist es möglich Kommunikation zwischen Echtzeit-Tasks und User-Space-Programmen zu betreiben.

Das Modul „`rtai_comedi.ko`“ ist ein Teil der Schnittstelle zwischen RTAI und Comedi. Mit ihr ist es möglich die Echtzeit-Comedi-Funktionen in einem Echtzeit-Modul auszuführen.

4.4.4 LXRT

LXRT kann in Abhängigkeit des RTAI-Moduls „`rtai_lxrt.ko`“ genutzt werden. Mit LXRT ist es möglich Echtzeitprogramme im User-Space auszuführen.

Dies hat folgende Vorteile:

- Fehler eines Echtzeitprogramms können sich im User-Space nicht gravierend auf die Stabilität das System auswirken.
- Das Debugging von User-Space-Programmen ist mit einfachen Debuggern besser möglich.

Aufgrund dieser Vorteile eignet sich LXRT besonders dazu spätere Echtzeit-Programme zu entwickeln und zu testen. Ist das Echtzeit-Programm ausreichend stabil, kann es als Kernel-Modul geladen werden und LXRT wird nicht mehr benötigt.
Wichtige Echtzeit-Programme sollten als Kernel-Modul ausgeführt werden, da LXRT den Nachteil hat, wegen der vermehrten Kontextwechsel, die Latenzzeit des Systems zu erhöhen.

4.4.5 RTAI-Lab

RTAI-Lab ist ein Programm, welches mit RTAI ausgeliefert wird. RTAI-Lab wird im User-Space ausgeführt und kann über FIFOs mit den Echtzeit-Tasks kommunizieren.

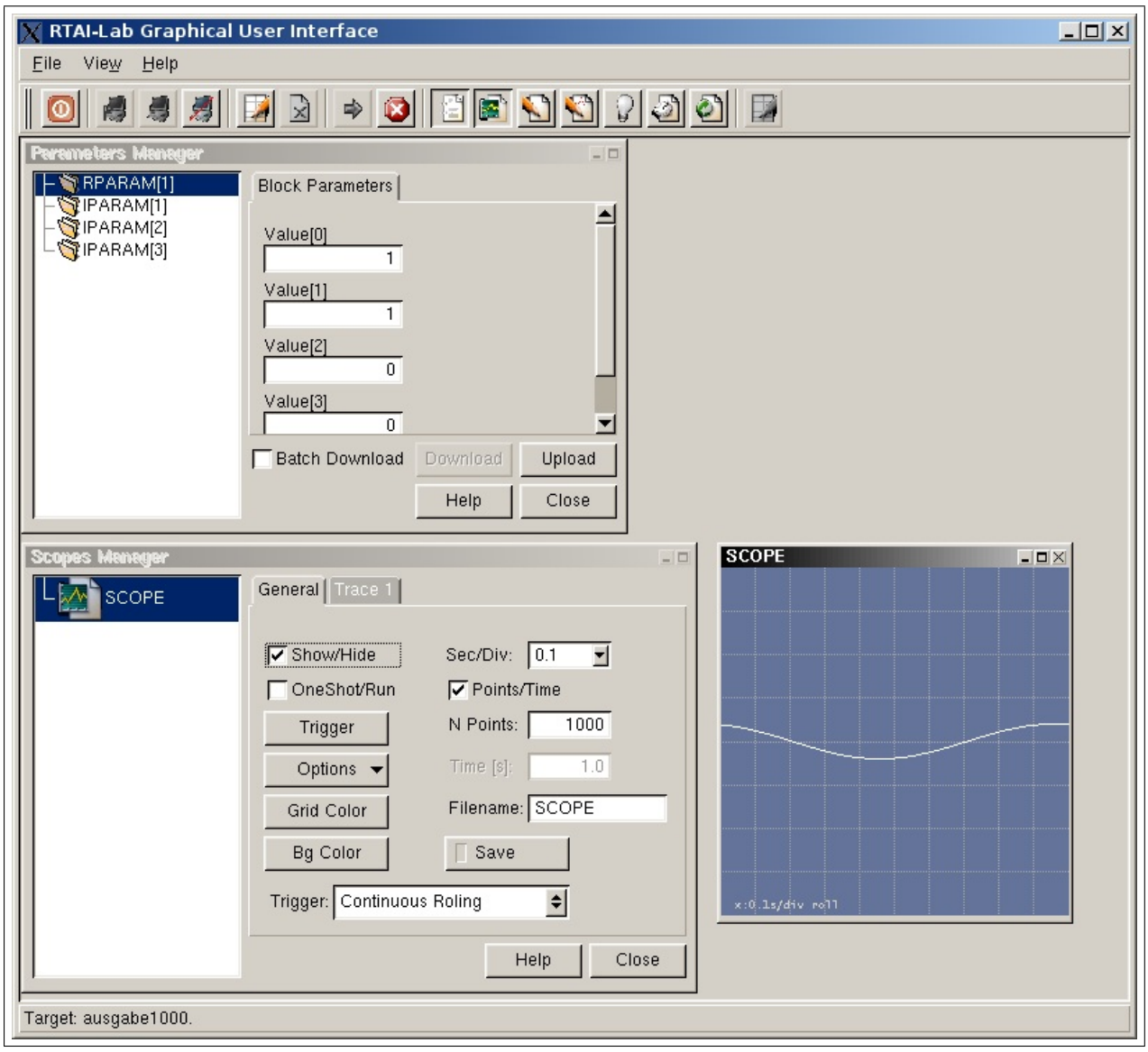

Abbildung 4.7: RTAI-Lab-GUI mit „Parameter Manager“ und „Scopes Manager“

Abbildung 4.7 zeigt die GUI von RTAI-Lab. Mit RTAI-Lab kann ein „Scope“ und ein „Meter“ neben diversen LEDs angezeigt werden. Wurden Echtzeitprogramme für RTAI kompiliert, so können viele Informationen mit RTAI-Lab graphisch dargestellt werden. RTAI-Lab verfügt auch über einen „Parameters Manager“ mit dem diverse Parameter und Variablen eines Echtzeit-Kernel-Moduls zur Laufzeit verändert werden können. Das RTAI-Lab ist das Analogon zur LabVIEW Steuer-GUI.

Damit RTAI-Lab für ein Echtzeit-Programm, welches im Kernel-Space läuft, genutzt werden kann, muss das Echtzeitprogramm nach einem bestimmten Standard programmiert worden sein. Dieser Standard wird von den Programmen „Scilab/Scicos“ und „Matlab/Simulink“ unterstützt.
Scilab/Scicos und Matlab/Simulink sind gleichartige Programme zur Erstellung von Kernel-Modulen speziell für RTAI. Scilab/Scicos ist die Open-Source-Veriante von Matlab/Simulink.

4.5 Scilab/Scicos

„Scilab“ ist ein Softwarepaket, welches wissenschaftliche Programme enthält. Eines dieser Programme ist „Scicos“ der „Block Diagramm Modellierer/Simulator“. Mit Scicos können Programme in Form von Datenfluss-Darstellungen entworfen und kompiliert werden.
Mit der RTAI-Erweiterung von Scilab werden in Scicos zusätzliche RTAI-Elemente verfügbar. Zu diesen Elementen zählen beispielsweise FIFOs, das Scope aus RTAI-Lab und Comedi-Elemente für die Ausgabe von Signalen auf der Mess- und Steuerkarte.

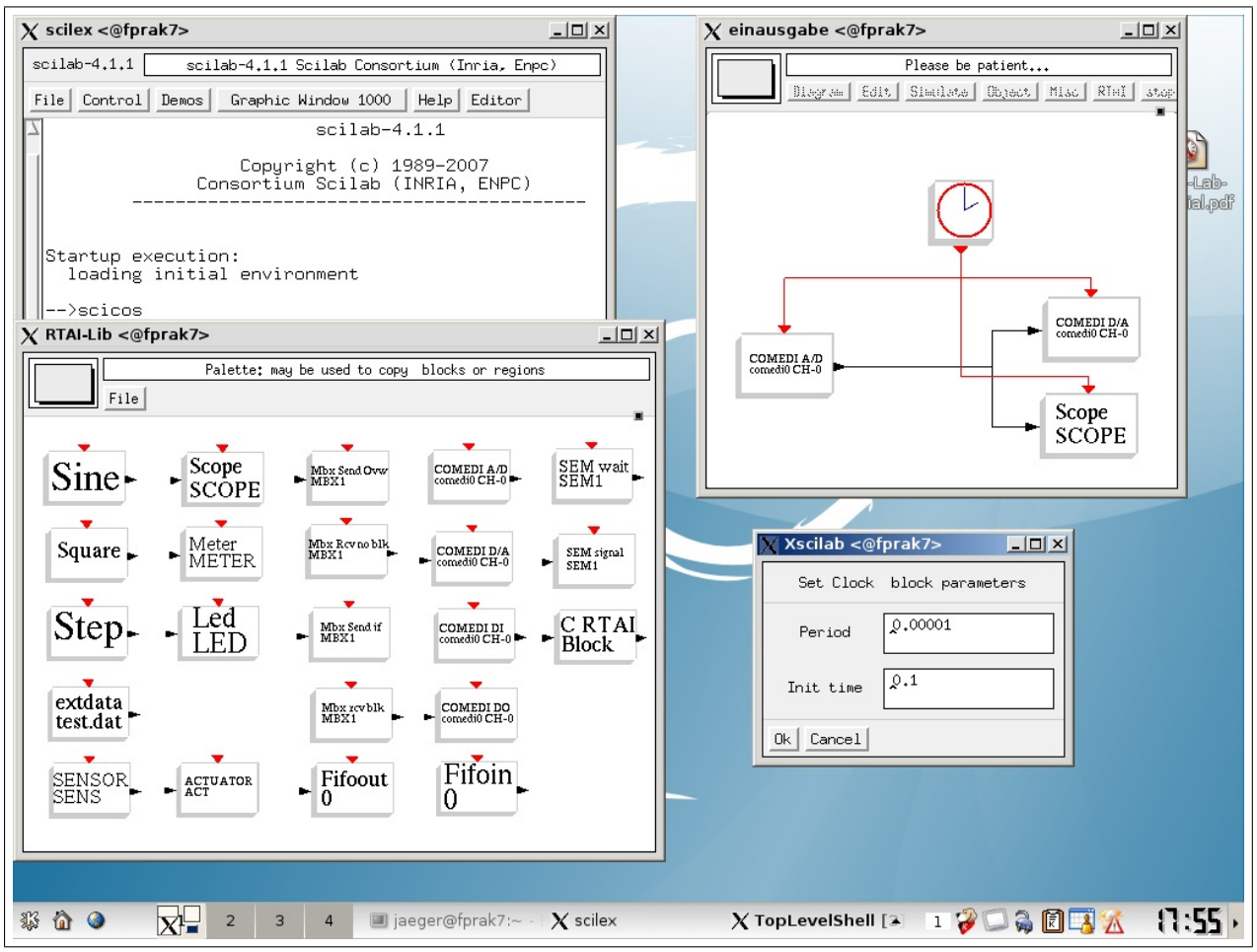

Abbildung 4.8: Datenfluss-GUI von Scicos

Abbildung 4.8 zeigt die Datenfluss-GUI von Scicos und die Palette aller RTAI-Elemente. Das dargestellte Scicos-Programm liest Werte über den analogen Eingang der Mess- und Steuerkarte (Device: comedi0) ein und gibt sie direkt über den analogen Ausgang aus. Zusätzlich werden die eingelesenen Werte auf dem RTAI-Lab Scope angezeigt.
Die schwarzen Linien der Datenfluss-GUI geben die Wege an auf denen Werte ausgetauscht werden. Die roten Linien verteilen das Interrupt-Signal, um eine Steuerperiodendauer auf die einzel-

nen Funktionen zu übertragen. In Abbildung 4.8 wurde die Steuerperiodendauer auf 0,00001 s festgelegt. Die Datenfluss-GUI von Scicos ist das Analogon zur LabVIEW Datenfluss-GUI.

Kapitel 5

Der Ansatz

Es gilt nun eine Regelung zu realisieren mit den zur Verfügung stehenden Mitteln. Zur Ein- und Ausgabe von analogen Signalen ist Comedi die einzig obligate Lösung, da Comedi direkt Echtzeit-Kommunikation mit der Mess- und Steuerkarte unterstützt und gleichzeitig mit einer beachtlichen Menge an Mess- und Steuerkarten verwendet werden kann.

Zur Umsetzung einer Regelung für das Experiment ergeben sich zwei Möglichkeiten:

Möglichkeit 1 (Regelung mit LabVIEW und Comedi):

Es ist möglich Comedi in LabVIEW zu integrieren. Dadurch kann eine Kommunikation mit der Mess- und Steuerkarte stattfinden und es können Werte mit LabVIEW verarbeitet werden. Diese Möglichkeit konnte in dieser Arbeit ohne Erfüllung der Echtzeitanforderung umgesetzt werden.

Möglichkeit 2 (Regelung mit RTAI, RTAI-Lab, Scilab/Scicos und Comedi):

Damit die Echtzeitanforderung im vollen Maß erfüllt wird, wurde die Kombination der Softwarepakete Scilab/Scicos mit RTAI-Lab und Comedi verwendet. Diese Möglichkeit basiert somit vollständig auf Open-Source-Produkten.

5.1 Regelung mit LabVIEW und Comedi

Die Software LabVIEW verfügt über das Konzept der „Shared Libraries“. Mit Shared Libraries ist es möglich extern programmierte Funktionen mit einem **G**NU-**C**-**C**ompiler (kurz GCC) zu kompilieren und diese als virtuelles Instrument in LabVIEW einzufügen. Dadurch können Comedi-Funktionen in LabVIEW eingebunden werden und zur Ein- oder Ausgabe von elektrischen Signalen genutzt werden.

Soll der Regelalgorithmus selbst mit LabVIEW umgesetzt werden, so geschieht dies allerdings nicht in Echtzeit. Wie in Kapitelpunkt 4.2 erwähnt wurde, ist LabVIEW nicht ohne Echtzeit-Modul und Target-System echtzeitfähig.

Eine andere Möglichkeit LabVIEW zu nutzen ist es den Regelalgorithmus in einem RTAI-Kernel-Modul zu isolieren und mit FIFOs Werte zwischen dem Regelalgorithmus und LabVIEW auszutauschen. Dies wäre ein analoger Ansatz, wie er mit RTAL-Lab und den Scicos-Programmen realisiert wurde. Dadurch kann ein stabiles RTAI-Kernel-Modul die Ausführung der Regelung übernehmen und es könnten die Vorteile aller benutzerfreundlichen LabVIEW GUIs genutzt werden. Bei dieser Möglichkeit ist es nicht nötig Comedi in LabVIEW einzubinden. Es müssten lediglich einfache Funktionen in LabVIEW, als Shared Library eingebunden werden, welche den Datenaustausch über die FIFOs durchführen. Dieser Ansatz wurde nicht in dieser Arbeit umgesetzt.

5.2 Regelung mit RTAI, RTAI-Lab, Scilab/Scicos und Comedi

Dies ist eine vollständige Open-Source-Möglichkeit, bei der die gegenseitigen Kompatibilitäten gewährleistet sind. RTAI unterstützt Comedi mit dem optionalen RTAI-Modulen „`rtai_comedi.ko`“ und „`kcomedilib.ko`“. Scicos kann mit der zusätzlichen RTAI-Palette ausgestattet werden, dadurch können die Comedi-Devices und andere RTAI-Funktionalitäten in der Datenfluss-GUI von Scicos genutzt werden.
Da somit alle RTAI-Funktionalitäten in den Scicos-Programmen genutzt werden ist es auch möglich RTAI-Lab zur benutzerfreundlichen Steuerung der Scicos-Programmen zu nutzen. Scicos ersetzt die Datenfluss-GUI von LabVIEW und RTAI-Lab ersetzt die Steuer-GUI von LabVIEW.

5.3 Gewählte Lösung

Die Möglichkeit „Regelung mit RTAI, RTAI-Lab, Scilab/Scicos und Comedi“ wurde für die Regelung des Experimentes in dieser Arbeit umgesetzt. Dadurch entstand eine Open-Source-Lösung, dessen Softwarepakete ständig aktualisiert werden können.
Aus didaktischen Gründen wurde auch der erste Teil der Möglichkeit „Regelung mit LabVIEW und Comedi“ umgesetzt. Das heißt es kann eine nicht-Echtzeit-Regelung mit LabVIEW und den nicht-Echtzeit-Funktionen von Comedi über die Shared Libraries vorgenommen werden.

Kapitel 6

Vorbereitung des Experiment-PC

In diesem Kapitel soll beschrieben werden, wie der Experiment-PC für das Experiment vorbereitet wurde. Dazu gehört die Installation und Konfiguration aller Softwarepakete auf den Experiment-PC.

Es wurden folgende grundlegende Softwarepakete auf dem Experiment-PC installiert:

- Gentoo-Linux. Alle Kernel-Konfigurationen wurden so ausgewählt, dass nur die Hardwarekomponenten des Experiment-PC unterstützt werden. Auf dem Gentoo-Linux wurde der ressourcensparende Fenstermanager „IceWM“ installiert.
- Nachträglich wurden der NVIDIA Grafikkartentreiber installiert. Es wurden keine Open-Source-Teiber benutzt sondern das Grafiktreiberpaket vom Hersteller NVIDIA
- Es wurden alle Comedi-Pakete installiert. Das heißt Comedi und Comedilib.
- Es wurde das RTAI-Erweitungspaket installiert.
- Zuletzt wurde Scilab mit Scicos installiert.

Gentoo-Linux ist eine sehr schlanke Linux-Distribution. Bei Gentoo-Linux werden nur Pakete installiert, deren Installation expliziert vom Benutzer durchgeführt wird. Dadurch kann das Linux klein und übersichtlich gehalten werden. Ein großer Vorteil von Gentoo-Linux ist, dass alle Pakete in Sourcecode erhältlich sind und so speziell für den Experiment-PC kompiliert werden konnten. Die Hardwarekomponente, welche den größten Einfluss auf den Kompilierungsprozess hatte ist der Prozessor des Experiment-PC. Jeglicher Sourcecode wurde somit für „Athlon-XP“ kompiliert.

6.1 Versionen von Softwarepaketen

Der Sinn der Softwarepakete des Experiment-PC soll sein, dass mit dem „emerge“-Tool von Gentoo-Linux jederzeit Updates vorgenommen werden können, ohne die Stabilität des Experiment-PC großartig zu beeinträchtigen. In manchen Fällen zeigte sich in der Praxis trotzdem eine, wenn auch dokumentierte, Versionen-Abhängigkeit.
Aus diesem Grund soll hier die Tabelle 6.1 mit allen verwendeten Softwarepaketen und den verwendeten Versionen folgen:

Software-Paket	**Version**	**Bemerkung**
Gentoo-Linux	2007.0	Bei Gentoo entspricht diese Angabe der Profilnummer
Linux-Kernel	2.6.19 und 2.6.21	Für die RTAI-Erweiterung wurde 2.6.19 benutzt, für den normalen Betrieb 2.6.21.
GNU-C-Compiler	4.1.1 und 3.3.6	Für die Gentoo-Installation wurde die GCC Version 4.1.1 verwendet, ab der RTAI-Erweiterung nur 3.3.6, da dies empfohlen wurde.
gcc-config	1.3.14	Zur Umstellung des standardmäßig verwendeten GNU C Compilers
ncurses	5.6-r1	Zur Darstellung des Kernelkonfigurationsmenüs.
automake	1.10	Zur Erstellung von „Makefile.in“-Dateien
autoconf	2.61	Zur Erstellung von Autokonfigurationsdateien
libtool	1.5.23b	Eine Shared Library
bison	2.2	
doxygen	1.5.2	Zur Erstellung von HTML-Dokumentationen für diverse Programmiersprachen
GTK	2.10.13	für grafische Benutzeroberflächen
Xorg-X11	7.2	X-Server
IceWM	1.2.30	Wurde mit Rox-Filer als Dateimanager und Firefox erweitert.
Rox-Filer	2.6	benutzter Dateimanager unter IceWM
Firefox	2.0.0.6	Web-Browser
NVIDIA-Treiber	100.14.09	Grafikkartentreiber für X
Comedi	0.7.74	siehe Kapitelpunkt 2.5.1
Comedilib	0.8.0	siehe Kapitelpunkt 2.5.1
RTAI	3.5 Vulcano	„Vulcano“ steht für die „stable“ Versionen. „Magma“ steht für die Development-Versionen
Mesa Library	6.5.2	Wird für eFLTK benötigt
eFLTK	2.0.5	Wird für RTAI-Lab unter X benötigt. Es wurde die CVS-Version verwendet.
LabVIEW	7.1	

Tabelle 6.1: Auflistung und Versionen wichtiger Software-Pakete auf dem Experiment-PC

Wenn in dieser Arbeit ein Paket auf dem Experiment-PC installiert wurde, so geschah dies in erster Linie mit dem „emerge“-Tool und wurde nur in Ausnahmefallen auf anderem Weg erledigt. Auf solche Ausnahmen wird in der Installationsbeschreibung hingewiesen. Es wurden prinzipiell keine experimentellen Pakete, sondern nur Pakete installiert, welche als „stable“ markiert wurden.

6.2 Installation des Gentoo-Linux-Grundsystem

Die grundlegende Installation von Gentoo-Linux wurde auf dem Experiment-PC nach dem Gentoo Handbuch [gentoohb] durchgeführt. Unter dem Inhaltspunkt „Gentoo installieren“ wird die Gentoo Installation in 10 Schritten vollzogen. Diese 10 Schritte entsprechen den Gliederungspunkten 2 bis 11 im Inhaltspunkt „Gentoo installieren“ des [gentoohb].
In den folgenden Kapitelpunkten wird nur auf Besonderheiten oder Abweichungen vom Gentoo Handbuch eingegangen.

6.2.1 Schritt 1: Auswählen des richtigen Installationsmediums

Das Gentoo auf dem Experiment-PC wurde mit der LiveCD durchgeführt. Es wurde der konsolenbasierende „Installer“ verwendet, um den größt möglichen Konfigurationseinfluss zu erhalten. Es wurde auf dem Experiment-PC nur eine stage3-Tarball Installation durchgeführt.

6.2.2 Schritt 2: Konfiguration des Netzwerks

Auf dem Experiment-PC wurde eine „Automatische Netzwerkkonfiguration“ (Gliederungspunkt 3.b. im [gentoohb]) mit einer festen IP-Adresse durchgeführt. Im Netzwerk des Experiment-PC wurde somit kein DHCP-Server verwendet.

6.2.3 Schritt 3: Vorbereiten der Festplatte(n)

Auf dem Experiment-PC wurde eine Festplatte verwendet. Damit das Gentoo-System mit höherer Datensicherheit benutzt werden kann, wurden die Hauptverzeichnisse von Gentoo (`/usr`, `/var`, `/opt`, `/home`, `/tmp`) auf unterschiedliche Partitionen verteil. Dies hat den Vorteil, dass bei Datei- oder Partitionsverlusten nur ein kleiner Teil wiederhergestellt werden musste. Datei- und Partitionsverluste sind während dieser Arbeit aufgetreten, siehe „RTAI-Lab-Absturz-Problem“ Kapitelpunkt 6.5.3.
Ein weiterer Vorteil von mehreren Partitionen ist, dass Festplatten-Checks nur auf bestimmen Partitionen ausgeführt werden müssen und so die benötigte Zeit für den Festplatten-Check minimiert wird.

Folgende Partitionen wurden auf der Festplatte des Experiment-PC erstellt (Angabe nach „$ cfdisk“):

Name	Flags	Part Type	FS Type	[Label]	Size (MB)
hda1	Boot	Primary	Linux ext2	[boot]	32.91
hda2		Primary	Linux swap/Solaris		2048.10
hda5		Logical	Linux ext2	[root]	98.71
hda6		Logical	Linux ReiserFS	[var]	2048.10
hda7		Logical	Linux ReiserFS	[opt]	501.75
hda8		Logical	Linux ReiserFS	[usr]	7164.22
hda9		Logical	Linux ReiserFS	[tmp]	1019.94
hda10		Logical	Linux ReiserFS	[home]	18556.24
hda4		Primary	Linux ext2		8587.20

Abbildung 6.1: Partitionenangabe von „cfdisk“ des Experiment-PC

Zur Übersicht folgt die Partitionenaufteilung (Abbildung 6.2) des Experiment-PC (Angabe nach „$ df --Th“):

Filesystem	Type	Size	Used	Avail	Use%	Mounted on
/dev/hda5	ext2	92M	81M	6.3M	93%	/
udev	tmpfs	505M	2.7M	503M	1%	/dev
/dev/hda6	reiserfs	2.0G	488M	1.5G	25%	/var
/dev/hda7	reiserfs	479M	160M	319M	34%	/opt
/dev/hda8	reiserfs	6.7G	3.3G	3.5G	49%	/usr
/dev/hda10	reiserfs	18G	1.7G	16G	10%	/home
/dev/hda9	reiserfs	973M	34M	940M	4%	/tmp
shm	tmpfs	505M	0	505M	0%	/dev/shm

Abbildung 6.2: Ausgabe von „df --Th“ des Experiment-PC

In Abbildung 6.1 ist zu erkennen, dass für alle Partitionen der Festplatte „/dev/hda“ Labels vergeben wurden. Zum Beispiel „hda8“ hat den Label „usr“. Diese Labels machen das „mounten“ von Partitionen intuitiver. Labels für „reiserfs“-Partitionen können mit dem Kommando:

```
reiserfstune -l /dev/hda8 usr
```

vergeben werden. Das obige Kommando zeigt beispielsweise die Festlegung des Labels „usr“ für die Partition „hda8“.

Zu Beachten 6.1 (Dateiverluste)

Auf dem Experiment-PC wurde hauptsächlich das Dateisystem „reiserfs 3.6“ benutzt. Aufgrund des „RTAI-Lab-Absturz-Problem“ sind immer wieder Dateiverluste auf der Partition „`hda8`“ aufgetreten. Die Dateiverluste waren hauptsächlich durch Verluste der „Inodes“ gekennzeichnet. Aus diesem Grund wurden die reiserfs-Partitionen grundsätzlich mit dem Parameter „`notail`“ gemountet. Dadurch werden keine Inodes verwendet. Der mount-Parameter wurde in der Datei „`/etc/fstab`“ ergänzt.

Die Datei „`/etc/fstab`“ des Experiment-PC Gentoo Systems:

```
# <fs>          <mountpoint>    <type>      <opts>                  <dump/pass>
LABEL=boot      /boot           ext2        noauto                  1 2
LABEL=root      /               ext2        atime                   0 1
/dev/hda2       none            swap        sw                      0 0
LABEL=var       /var            reiserfs    atime                   0 0
LABEL=opt       /opt            reiserfs    atime                   0 0
LABEL=usr       /usr            reiserfs    atime,notail            0 0
LABEL=home      /home           reiserfs    atime                   0 0
LABEL=tmp       /tmp            reiserfs    atime                   0 0
LABEL=backup    /backup         ext3        noauto                  0 0
/dev/cdrom      /mnt/cdrom      auto        noauto,ro,user,exec     0 0
/dev/fd0        /mnt/floppy     auto        noauto                  0 0
```

Abbildung 6.3: Die Datei „`/etc/fstab`“ des Experiment-PC

In der Spalte „`# <fs>`“ der Abbildung 6.3 ist zu erkennen, dass die Datei „`/etc/fstab`“ mit den festgelegten Labels jeder Partition arbeitet. Durch ein Label wird eine Partition identifiziert und dann in ein Mountpoint (Verzeichnis) eingehängt.
In den Mountpoint „`/backup`“ kann eine zweite Backup-Festplatte („`/dev/hdb`“) gemountet werden. Der Parameter „`notail`“ kann bei Bedarf bei den anderen reiserfs-Partitionen ergänzt werden.

6.2.4 Schritt 4: Installation der Gentoo Installationsdateien

In der Datei „`/etc/make.conf`“ wurden für das Gentoo und die Hardware des Experiment-PC spezifische Angaben gemacht. Aus diesem Grund folgen nun die Angaben der „`make.conf`“ des Experiment-PC.

```
USE="X radeon -gnome -kde -rage128 -sis -matrox -3dfx -gamma -i8x0 -alsa"
INPUT_DEVICES="mouse keyboard"
VIDEO_CARDS="nvidia"
CHOST="i686-pc-linux-gnu"
CFLAGS="-march=athlon-xp -O3 -pipe"
PORTAGE_ELOG_CLASSES="warn error"
MAKEOPTS="-j2"
```

Abbildung 6.4: Die Datei „/etc/make.conf“ des Experiment-PC

6.2.5 Schritt 5: Installation des Gentoo Basissystems

Zur Definition der Locales wurde für den Experiment-PC nur folgende Zeile eingetragen:

```
de_DE@euro ISO-8859-15
```

6.2.6 Schritt 6: Konfiguration des Kernels

Gliederungspunkt 7.c. Standard: Manuelle Konfiguration

Der Kernel des Experiment-PC Gentoo wurde mit dem Gliederungspunkt 7.c. im [gentoohb] manuell konfiguriert. Es wurden nur Hardwarekomponenten eingebunden, welche im Experiment-PC vorhanden sind. Bevorzugt wurden Kernel-Konfigurationen als Modul ausgewählt. Dies gibt die Möglichkeit die Kernel-Module später noch einmal nur bei Bedarf zu laden.
Der Linux-Kernel, welcher mit dem Schritt 6 konfiguriert und kompiliert wurde hat im Experiment-PC die Version 2.6.21 und gehört zum Standardsystem des Experiment-PC. Das Standardsystem ist nicht in der Lage Echtzeit mit RTAI umzusetzen. Für die RTAI-Erweiterung musste ein neuer Kernel 2.6.19 angepasst und kompiliert werden. Mehr dazu im Kapitelpunkt 6.4.

Gliederungspunkt 7.e. Kernelmodule

Für das Gentoo-Standardsystem des Experiment-PC wurden zwei Module in der Datei „/etc/modules.autoload.d/kernel-2.6“ eingetragen. Diese sind:

```
via-agp
agpgart
```

Abbildung 6.5: Die Datei „/etc/modules.autoload.d/kernel-2.6“ des Experiment-PC

6.2.7 Schritt 9: Konfiguration des Bootloaders

Auf dem Experiment-PC wurde der Bootloader „Grub“ ohne Framebuffer installiert. Nach der Installation der RTAI-Erweiterung wird ein neuer, für RTAI präparierte, Linux-Kernel zur Verfügung stehen. Diese neue echtzeitfähige Kernel kann dann mit Grub angebootet werden.

6.3 Installation/Einrichtung des Fenstermanagers „IceWM“

Um den Fenstermanager IceWM in Betrieb zu nehmen, wurde die Installation und Konfiguration weiterer Pakete vorgenommen. Die Installation der Pakete wurde ausschließlich mit dem „emerge“-Tool vorgenommen.

X-Server

Es wurde der X-Server mit dem folgenden Befehl installiert:

```
$ emerge --verbose xorg-x11
```

IceWM

Der eigentliche Fenstermanager IceWM wurde unter Gentoo mit folgendem Befehl installiert:

```
$ emerge --verbose icewm
```

Rox-Filer

Rox-Filer ist ein kleiner Dateimanager, der sich in den Festermanager IceWM einbinden lässt. Zur Installation des Rox-Filer wurde folgender Befehl verwendet:

```
$ emerge --verbose rox
```

Nähere Informationen zur Installation des Rox-Filer unter Gentoo-Linux werden auf der Homepage http://rox.sourceforge.net/desktop/Gentoo bereitgestellt.

Damit Rox-Filer standardmäßig beim Start von IceWM benutzt wird, wurde für jeden Benutzer die Zeile:

```
rox -p Default -b Default &
```

an die Datei „`/home/[Benutzer]/.xinitrc`“[1] angehängt.

[1] Dabei steht der Ausdruck `[Benutzer]` für die bezeichnende Zeichenkette eines Benutzers

Grafikkartentreiber

Im Experiment-PC wurde eine NVIDIA Geforce FX 5200 Grafikkarte installiert. Mit dieser Grafikkarte konnte das „Grafikkarten-Problem“, siehe Kapitelpunkt 6.5.2, behoben werden. Es wurde der originale Herstellertreiber von NVIDIA, nach den „Gentoo-Linux nVidia Guide“ [nvidiaguide], installiert.
Nach der Installation des Grafikkartentreibers war mit dem Befehl „`$ glxinfo`“ zu erkennen, dass das „direct rendering“ nicht aktiviert war (`direct rendering: No`). Der Grund dafür lag an den eingeschränkten Zugriffsrechten auf den Grafikkarten-Devices „`/dev/nvidia0`“ und „`/dev/nvidiact1`“.
Die korrigierten Zugriffsrechte konnten mit einer Änderung der Datei „`/etc/modprobe.conf`“ hergestellt werden. In dieser Datei existiert die Zeile:

```
options nvidia NVreg_DeviceFileMode=432 NVreg_DeviceFileUID=0
NVreg_DeviceFileGID=27 NVreg_ModifyDeviceFiles=1
```

Dieser Zeile legt die Zugriffsberechtigungen auf die Grafikkarten-Devices mit „`NVreg_DeviceFileMode=432`“ fest. Wird dies in „`NVreg_DeviceFileMode=0666`“ geändert, so ist nach einem Experiment-PC Neustart das Direct Rendering aktiviert.

6.4 Installation und Konfiguration von Comedi, RTAI und Scilab/Scicos

In Kapitelpunkt 6.2 bis 6.3 wurde die Installation des Gentoo-Linux Standardsystems besprochen. Aufbauend auf diesem Standardsystem gilt es nun die RTAI-Erweiterung, inklusive des Comedi-Interface zur Mess- und Steuerkarte, auf den Experiment-PC einzurichten. Ist der Experiment-PC danach bereit RTAI-Anwendungen auszuführen wird Scilab/Scicos installiert.
Alle notwendigen Installationen und Konfigurationen, in diesem Kapitelpunkt, wurden nach dem RTAI-Lab Tutorial [rtaitut] durchgeführt. Im [rtaitut] steht das Ziel RTAI-Lab in Verbindung mit Scilab/Scicos und einer Comedi unterstützten Mess- und Steuerkarte einzusetzen. Das [rtaitut] setzt nach einer erfolgreichen Linux-Standardsystem-Installation an.

Der Gliederungspunkt 2 (Installation) des [rtaitut] erläutert Schritt für Schritt die Installation und Konfiguration von Comedi, RTAI und Scilab/Scicos. In den folgenden Kapitelpunkten unter Kapitelpunkt 6.4 dieser Arbeit werden nur vom [rtaitut] abweichende Schritte erläutert.

6.4.1 Gliederungspunkt 2.1.2: Software requirements

Laut [rtaitut] wurde zum Zeitpunkt dieser Arbeit die Verwendung eines GCC mit der Version kleiner 4.0 empfohlen. Aus diesem Grund wurde, in dieser Arbeit, zusätzlich der GCC 3.3.6 installiert und von nun an permanent verwendet.
Es wurde also das Gentoo-Linux Standardsystem des Experiment-PC mit dem GCC 4.1.1 kompiliert und ab Beginn der Benutzung des [rtaitut] der GCC 3.3.6 verwendet.

6.4.2 Gliederungspunkt 2.4: Linux kernel and RTAI patch

Es wurde Gliederungspunkt „Kernel 2.6.xx“ ausgeführt, da auf dem Experiment-PC der Linux-Kernel 2.6.19 für die RTAI-Erweiterung verwendet wird.

6.4.3 Gliederungspunkt 2.8: Comedi

3. Schritt

Comedi wurde im 3. Schritt mit kbuild („`--enable-kbuild`“) konfiguriert. Dadurch erhält der Benutzer später die Möglichkeit eigene Kernel-Module mit dem „kbuild“-Tool zu kompilieren.

4. Schritt

In diesem Schritt wird Comedi mit dem Aufruf „`$ make`“ kompiliert. Dadurch werden ebenfalls alle Treiber für alle unterstützten Mess- und Steuerkarten kompiliert. Damit die ME-4660i mit dem zugehörigen Comedi-Treiber angesprochen werden kann, musste der Comedi-Treiber verändert werden. Nähere Informationen dazu befinden sich in Kapitelpunkt 6.5.1 („ME-4660i-Problem“).

6.4.4 Gliederungspunkt 2.10: Scilab/Scicos

Unter Linux Gentoo gibt es die Möglichkeit Scilab mit dem „emerge“-Tool zu installieren. Dies wurde in dieser Arbeit nicht durchgeführt. Stattdessen wurde die Source-Version von Scilab benutzt und manuell kompiliert, so wie es laut [rtaitut] beschrieben wird. Dieses Vorgehen gewährleistet, dass von hier an folgende Ausführungen des [rtaitut] auf dem Experiment-PC analog ausgeführt werden konnten.

6.4.5 Gliederungspunkt 2.13: Load the modules

Um die Echtzeitfähigkeit des Experiment-PC herzustellen müssen nun die RTAI-Module geladen werden. Abweichen vom Gliederungspunkt 2.13 des [rtaitut] wird dieser Prozess auf dem Experiment-PC mit dem Tool „modprobe“ durchgeführt.
Damit „modprobe“ für diese Aufgabe verwendet werden kann, müssen die kompilierten RTAI-Module in das „modules“-Verzeichnis des Gentoo-Linux kopiert werden. Folgende Schritte wurden zusätzlich auf dem Experiment-PC als Administrator ausgeführt.

```
$ cd /usr/realtime/modules/
$ mkdir /lib/modules/2.6.19-rtai/rtai
```

Kopieren der kompilierten RTAI-Modulen in das „modules“-Verzeichnis des Gentoo-Linux:

```
$ cp * /lib/modules/2.6.19-rtai/rtai
```

Durch diese Ausführungen können alle benötigten Module auf dem Experiment-PC mit dem folgenden Skript geladen werden.

```
#!/bin/sh

modprobe rtai_hal
modprobe comedi
modprobe comedi_fc
modprobe ni_pcimio
modprobe rtai_up
modprobe kcomedilib
modprobe rtai_shm
modprobe rtai_comedi
modprobe rtai_mbx
modprobe rtai_netrpc
modprobe rtai_msg
modprobe rtai_sem
modprobe rtai_fifos
```

Abbildung 6.6: Skript zum Laden der RTAI-Module auf dem Experiment-PC

6.4.6 Konfiguration der ME-4660i während des Bootens

Mess- und Steuerkarten-Modul laden

Für das Gentoo RTAI-System des Experiment-PC wurde ein weiteres Modul in der Datei „`/etc/modules.autoload.d/kernel-2.6`“ eingetragen. Dieses ist das „me4000“-Modul. Dadurch wird der Treiber für die Mess- und Steuerkarte des Experiment-PC beim Booten des Gentoo-Linux geladen und alle nötigen Comedi-Devices erstellt.

FPGA der ME-4660i konfigurieren

Der FPGA der ME-4660i ist für deren korrekte Funktionsweise unverzichtbar. Ein FPGA hat die Eigenschaft bei Stromausfall seine Konfiguration zu verlieren und kann damit nicht mehr seine logischen Schaltungen ausführen. Bei jedem Neustart des Experiment-PC muss also der FPGA der ME-4660i neu konfiguriert werden. Dies wird mit einer Firmware über die Kommandozeile durchgeführt.
Zur Automatisierung dieses Vorgangs kann die Datei „`/etc/conf.d/local.start`“ angepasst werden. Diese Datei wird bei jedem Bootvorgang automatisch ausgeführt. Zur Konfiguration des FPGA auf der ME-4660i werden der „`local.start`“ folgende Zeilen angefügt:

```
/usr/local/sbin/comedi_config /dev/comedi0 me4000 -i /usr/local/src/comedi_nonfree_firmware_2005_03_19/me4000/me4000_firmware.bin
chmod 777 /dev/comedi0
```

Die erste Zeile lädt die Firmware „`me4000_firmware.bin`“ auf die ME-4660i und der Aufruf „`chmod 777 /dev/comedi0`“ setzt die Zugriffsrechte des neuen Comedi-Devices so, dass alle Benutzer Zugriff auf die Mess- und Steuerkarte erhalten.

6.5 Probleme bei der Vorbereitung

Nach Durchführung aller Schritte unter dem Kapitelpunkt 6.4 ist der Experiment-PC bereit Echtzeit-Programme mit Scilab/Scicos zu erstellen und diese auszuführen. In diesem Kapitelpunkt sollen alle Probleme erläutert werden, welche bei der Vorbereitung des Experiment-PC in dieser Arbeit aufgetreten sind. Ebenfalls wird die gewählte Lösung besprochen.

6.5.1 „ME-4660i-Problem“

Comedi unterstützt mit seiner Treiberdatenbank viele Mess- und Steuerkarten verschiedenster Firmen. Für den Experiment-PC ist nur ein einziger Treiber interessant. Die relevante Treiber-Datei ist, nach der Comedi-Installation, die Datei „`/usr/local/src/comedi/comedi/drivers/me4000.c`“. Dieser Treiber unterstützt, neben anderen Mess- und Steuerkarten der Firma Meilhaus, die ME-4660i. Leider funktioniert dieser Treiber nicht.

Der original Comedi-Treiber für Meilhaus-Karten besitzt zwei grundlegende Fehlfunktionen:

1. Die Anzahl der Subdevices wird für die ME-4660i nicht korrekt angegeben.
2. Die Firmware, zur Konfiguration des FPGAs der ME-4660i, kann nicht auf die Mess- und Steuerkarte geladen werden.

Lösung des „ME-4660i-Problem“

Als Folge einer früheren Arbeit wurden diese zwei Fehlfunktionen bereits für die Comedi Version 0.7.70 behoben. Da keine tiefgreifenden Änderungen des Meilhaus-Treibers zur Comedi Version 0.7.74 durchgeführt wurden, konnten die Korrekturen in dieser Arbeit identisch übernommen werden[2].

Bei den Korrekturen des ME4000-Treibers (me4000.c) handelt es sich um folgende Änderungen:

```
/* Changed wrong entry for ME-4600* by Jäger/Galvosas August 2007 */

static const me4000_board_t me4000_boards[] = {
{ ME-4660", 0x4660, { 2, 0 }, { 16, 0, 0, 0 }, { 4 }, { 3 } },
{ ME-4660i", 0x4661, { 2, 0 }, { 16, 0, 0, 0 }, { 4 }, { 3 } },
{ ME-4660s", 0x4662, { 2, 0 }, { 16, 8, 0, 0 }, { 4 }, { 3 } },
{ ME-4660is", 0x4663, { 2, 0 }, { 16, 8, 0, 0 }, { 4 }, { 3 } },
};
```

Dieses Array legt, nach der Korrektur, die korrekte Anzahl der Subdevices der ME-4660i fest. Es wurde festgestellt, dass dieses Array innerhalb des Treibers nicht verwendet wird. Trotz einer absichtlichen Falschangabe von Subdevices konnte das Tool „`board_info`“ die korrekten Subdevices der ME-4660i erkennen.

```
/*Change prototype to make it work Jäger/Galvosas/Petriconi Aug 2007*/

/* static int xilinx_download(comedi_device *dev); */
static int xilinx_download(comedi_device *dev, unsigned char *firmware,
int firmware_size);
```

Der Funktion „`xilinx_download()`“ können nun zwei zusätzliche Parameter übergeben werden. Dies ist notwendig, um die Firmware der ME-4660i auf den FPGA der Mess- und Steuerkarte zu laden.

```
/* two variables added for xilinx download Jäger/Galvosas/Petriconi Aug
2007 */

unsigned char *firmware;
int firmware_size;
```

[2] Wir danken Silvio Petriconi für seine Hilfe bei der Lösung dieses Problems.

```
/* Changed to make xilinx download work Jäger/Galvosas/Petriconi Aug
2007 */

/* result = xilinx_download(dev); */
firmware = comedi_aux_data(it->options, 0);
firmware_size = it->options[COMEDI_DEVCONF_AUX_DATA_LENGTH];
result = xilinx_download(dev, firmware, firmware_size);
```

```
/* Changed to make xilinx download work Jäger/Galvosas/Petriconi Aug.
2007 */

/* static int xilinx_download(comedi_device *dev){ */
static int xilinx_download(comedi_device *dev, unsigned char
*xilinx_firm, int xilinx_firm_size) {
```

Dies sind ebenfalls Änderungen, welche das Aufspielen der Firmware ermöglichen.

Durch die oben genannten Änderungen ist die korrekte Funktionalität der Mess- und Steuerkarte ME-4660i unter Comedi gewährleistet.

6.5.2 „Grafikkarten-Problem“

Das „Grafikkarten-Problem“ wurde, auf dem Experiment-PC, während der Arbeit mit RTAI-Lab festgestellt. Zu Beginn der Arbeit mit RTAI-Lab war im Experiment-PC eine ATI Radeon 7000 Grafikkarte installiert. Mit dieser Grafikkarte wurden die grafisch, dynamischen RTAI-Lab Elemente wie das „Scope“ oder das „Meter“ nicht ordnungsgemäß dargestellt. Bei der Darstellung dieser RTAI-Lab Elemente waren nur wenige Linien zu erkennen, so dass diese Elemente nicht benutzt werden konnten.
Zusätzlich kam es bei „Resize“-Versuchen der grafischen Elemente zum Einfrieren des Experiment-PC. Mit „Resize“-Versuchen sind Änderungen der Fenstergrößen gemeint.
Diese ATI-Fehler waren sowohl beim ATI-Open-Source-Treiber aus auch beim originalen ATI-Hersteller-Treiber vorhanden.

Lösung des „Grafikkarten-Problem“

Eine Lösung des Problems war der Wechsel der verwendeten Grafikkarte von ATI zu NVIDIA. RTAI-Lab konnte auf dem Experiment-PC erfolgreich mit den Grafikkarten NVIDIA Geforce FX 5700LE und NVIDIA Geforce FX 5200 getestet werden. Für die dauerhafte Nutzung wurde die FX 5200 gewählt, da diese passiv gekühlt war.

6.5.3 „RTAI-Lab-Absturz-Problem“

Unter Scilab können Echtzeitprogramme so programmiert werden, dass sie periodisch, mit einer bestimmten Periodendauer ausgeführt werden. Für die Grenzen der minimal wählbaren Periodendauer gelten die Grundlagen aus Kapitelpunkt 3.2.2 (Regelung in Echtzeit). Wird nun die Periodendauer zu kurz gewählt, so kann es bei der Ausführung des Scilab-Programms dazu kommen, dass der Linux-Echtzeit-Task nicht mehr ausgeführt wird und der Experiment-PC nur mit der Ausführung des Echtzeit-Programms beschäftigt ist. Dieser Fall könnte durch die Verteilungen 2 oder 3 aus dem Kapitelpunkt 3.2.2 ausgelöst werden.
Wenn der Linux-Echtzeit-Task nicht mehr ausgeführt werden kann, also keine Rechenzeit auf der CPU bekommt, kann das Echtzeit-Programm nicht mehr beendet und der Experiment-PC nicht mehr heruntergefahren werden. Der Experiment-PC muss dann durch Reset neu gestartet werden. Abhilfe bringt dann nur ein schnelleres Computersystem. Bei einem schnelleren Computersystem verkürzt sich die Latenz- und Berechnungszeit des Echtzeit-Programmalgorithmus, so dass die Reaktionszeit kürzer wird.

Durch das „Reseten“ des Experiment-PC, in diesem Zustand, entstand ein neues Problem. Im Augenblick des Resetens kam es zu enormen Dateiverlusten des „reiserfs“-Dateisysems, vor allem auf der Partition „`/dev/hda8`“. Diese Dateiverluste, wenn einmal aufgetreten, konnten nicht mit einem herkömmlichen „reiserfsck“-Partitionen-Check behoben werden und mussten mit einer Wiederherstellung des Gentoo-Linux behoben werden. Um die Wiederherstellung durchzuführen musste trotzdem ein Partitionen-Check ausgeführt werden, da sonst immer wieder Fehler beim Schreiben auf die Partition auftraten. Während eines Check war festzustellen, dass hauptsächlich die Inodes des „reiserfs“-Dateisystems verloren gegangen sind.

Eindämmung des „RTAI-Lab-Absturz-Problem“

Die Dateiverluste konnten eingedämmt werden indem, wie unter Kapitelpunkt 6.2.3 (Schritt3: Vorbereiten der Festplatte(n)) beschrieben, die betroffene „reiserfs“-Partition mit „`notail`“ gemountet wird.

Kapitel 7

Ergebnisse

In diesem Kapitel werden alle Ergebnisse dieser Arbeit zusammenfassend vorgestellt und erläutert. Zur Prüfung der Echtzeitfähigkeit des Experiment-PC wurden die von RTAI mitgelieferten Tools benutzt. Diese Tool befinden sich nach Installation der RTAI-Erweitungen im Verzeichnis „`/usr/realtime/testsuite`“.
Um die Echtzeitfähigkeit des Experiment-PC während der Benutzung der Mess- und Steuerkarte zu testen, wurde Scilab/Scicos verwendet.

7.1 Erreichte Ziele und Grenzen der Arbeit

Folgende Ziele wurden mit dieser Arbeit erreicht:

Ziel 1: *Die Software LabVIEW konnte in dieser Arbeit soweit erweitert werden, dass es mit ihr möglich ist Signale über die Mess- und Steuerkarte einzulesen oder auszugeben. Um dieses Ziel umzusetzen konnte das Softwarepaket Comedi verwendet werden. Mit Comedi ist es gelungen eine Mess- und Steuerkarte, welche nicht von der Firma National Instruments angeboten wird, unter LabVIEW zu benutzen.*

Ziel 2: *Es konnte der Experiment-PC mit der Echtzeiterweiterung „RTAI“ unter Gentoo-Linux ausgerüstet werden. Mit RTAI ist es nun möglich Programme unter „harten“ Echtzeitbedingungen auszuführen.*

Ziel 3: *Mit dem Softwarepaket Comedi ist es gelungen die Ansteuerung der Mess- und Steuerkarte ME-4660i durch Echtzeitprogramme zu bewerkstelligen.*

Ziel 4: *Mit Scilab/Scicos und RTAI-Lab konnte in dieser Arbeit ein Open-Source-Ersatz zur Software LabVIEW beschafft werden. Scilab/Scicos und RTAI-Lab können auf dem Experiment-PC benutzt werden um Echtzeitprogramme zu erstellen und zu steuern. Mit der Umsetzung dieses Ziel ist es nun möglich Echtzeit-Regelungen mit dem Experiment-PC durchzuführen.*

Folgende Grenzen wurden mit dieser Arbeit erkannt und nicht überschritten:

- *Die Kommunikation zwischen LabVIEW und der verwendeten Mess- und Steuerkarte ME-4660i konnte nicht in Echtzeit bewerkstelligt werden.*
- *Die Erstellung eines Regelalgorithmus für das Experiment, der mit dem Experiment-PC ausgeführt wird, wurde in dieser Arbeit nicht ausgeführt.*

7.2 Tests des RTAI Gentoo-Linux auf dem Experiment-PC

In diesem Kapitelpunkt soll der, mit dieser Arbeit vorbereitete, Experiment-PC auf seine Echtzeitfähigkeit getestet werden.

Durch die Tests können folgende Eindrücke und Einschätzungen gewonnen werden:

- Wie gut ist der Experiment-PC für Echtzeit geeignet?
- Wie groß sind die Latenzzeiten des Experiment-PC?
- Was ist die theoretische minimale Regelperiodendauer und damit die maximale Regelfrequenz für Echtzeitprogramme im allgemeinen?
- Welche maximale Regelfrequenz kann der Experiment-PC bei Verwendung der Mess- und Steuerkarte verkraften?
- Wo steht der Experiment-PC im Vergleich zu anderen RTAI Linux Systemen?

7.2.1 Interrupt-Latenzzeiten

Die Interrupt-Latenzzeiten sind spezielle Latenzzeiten, welche gemessen werden können indem ein Interrupt ausgelöst wird. Die verstrichene Zeit bis der Echtzeit-Task auf den Interrupt reagiert und seine Berechnungen fortsetzt wird Interrupt-Latenzzeit genannt. Solche Interrupts werden von Timern beispielsweise in Scilab/Scicos Echtzeit-Programmen genutzt, um einem Echtzeit-Task periodisch ablaufen zu lassen.

7.2.1.1 im Kernel-space

Im Verzeichnis „`/usr/realtime/testsuite/kern`“ werden alle Testprogramme gespeichert, welche im Kernel-Space ausgeführt werden. Die Interrupt-Latenzzeiten des Experiment-PC wurden mit einem RTAI-Tool gemessen, welches sich im Verzeichnis „`/usr/realtime/testsuite/kern/latency`“ befindet. Ein Administrator, der sich in diesem Verzeichnis befindet und alle RTAI-Module geladen hat, kann mit „`$ inmod latency_rt.ko timer_mode=1`“ ein Echtzeit-Programm als Kernelmodul laden, welches die Interrupt-Latenzzeiten ermittelt. Das Echtzeit-Programm wird nun im Kernel-Space ausgeführt und übermittelt die gemessenen Latenzzeiten über eine FIFO in den User-Space. Mit dem Aufruf „`./display`“ können die Latenzzeiten auf der Konsole ausgegeben werden.

Periodischer Interrupt-Latenzzeiten

Die periodischen (`timer_mode=1`) Interrupt-Latenzzeiten des Experiment-PC sind:

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is periodic

 RTAI Testsuite - KERNEL latency (all data in nanoseconds)
 RTH|   lat min|   ovl min|   lat avg|   lat max|   ovl max|   overruns
 RTD|     -3872|     -3872|      -275|      3206|      3206|   0
 RTD|      -764|     -3872|      -275|       268|      3206|   0
 RTD|     -3611|     -3872|      -275|      3485|      3485|   0
 RTD|     -1508|     -3872|      -275|       965|      3485|   0
 RTD|      -798|     -3872|      -275|       625|      3485|   0
 RTD|      -903|     -3872|      -275|       307|      3485|   0
 RTD|     -6159|     -6159|      -275|      5737|      5737|   0
 RTD|     -5585|     -6159|      -275|      5393|      5737|   0
 RTD|     -4410|     -6159|      -275|      3981|      5737|   0
 RTD|      -884|     -6159|      -275|       285|      5737|   0
```

Abbildung 7.1: periodische Latenzzeit-Messung im Kernel-Space auf dem Experiment-PC

Funktionsweise des Latenzzeit-Programms

Als Kern des Programms wird solange gewartet bis nach einer Periode von 100000 ns ein Interrupt durch einen Timer ausgelöst wird. Dann wird im Zeitschritt n die Systemzeit t_n in Nanosekunden ermittelt und die Latenzzeit berechnet.

$Latenzzeit = t_n - t_{n-1} - Periode$

Die Ausgabe einer neuen RTD Zeile wird dann ausgeführt, wenn 1 s vergangen ist. Innerhalb des Latenzzeit-Programms wird also die Latenzzeitmessung in einer Schleife so oft ausgeführt bis eine Sekunde Echtzeit vergangen ist.

Bezeichnung	Erklärung
lat min	Gibt die kürzeste Latenzzeit der zuletzt gemessenen Sekunde an.
ovl min	Gibt die kürzeste Latenzzeit an, die nach Start des Latenzzeit-Programms gemessen wurde.
lat avg	Gibt den Durchschnitt aller Latenzzeiten der zuletzt gemessenen Sekunde an.
lat max	Gibt die längste Latenzzeit der zuletzt gemessenen Sekunde an.
ovl max	Gibt die längste Latenzzeit an, die nach Start des Latenzzeit-Programms gemessen wurde.
overruns	Gibt an wie oft die Timerperiode nicht eingehalten werden konnte. Sollte sich dieser Wert erhöhen, so ist dies ein Indiz dafür, dass das Computersystem nicht für Echtzeitprogramme mit der vorgegebenen Periode geeignet ist, da schon die Latenzzeit zu groß ist.

Tabelle 7.1: Spalten des Latenzzeit-Programms

Interpretation der Ergebnisse

Die längste Latenzzeit aus der Abbildung 7.1 beträgt 5737 ns. Die Spalte „`ovl max`“ gibt Auskunft über die längste aufgetretene Latenzzeit. Besonders für harte Echtzeit ist diese Spalte besonders wichtig.
Der Experiment-PC könnte also theoretisch eine maximale Regelfrequenz von rund 170 kHz erreichen. In der Praxis benötigt der Regelalgorithmus zusätzliche Rechenzeit. Diese Rechenzeit senkt die maximale Regelfrequenz herab. Da der Regelalgorithmus nicht implementiert wurde, kann nicht angegeben werden mit welcher theoretischen Frequenz der Regelalgorithmus ausgeführt werden kann. Die theoretische maximale Frequenz des Regelalgorithmus liegt allerdings unter 170 kHz.

Es ist in Abbildung 7.1 festzustellen, dass negative Latenzzeiten gemessen wurden. Dies zeigt vor allem die „`lat min`“ Spalte. Die negativen Latenzzeiten kommen dadurch zustande, dass $(t_n - t_{n-1}) < Periode$. Es kann also auch vorkommen, dass eine neue Systemzeit gemessen wird, bevor die eigentliche Periodenzeit verstrichen wird. Dies ist allerdings ungefährlich, da die Periodenzeit nicht überschritten wird.

Oneshot Interrupt-Latenzzeiten

„oneshot“ (einmalige) Interrupt-Latenzzeiten können mit dem Latenzzeit-Programm ebenfalls gemessen werden. Dazu genügt der Ausruf „`$ inmod latency_rt.ko`“. Mit „`./display`“ können wieder die Ausgaben auf der Konsole angezeigt werden.
Bei der Messung der Oneshot Interrupt-Latenzzeiten wird der Timer nicht auf einen periodischen Interrupt programmiert, sondern muss nach jeder Periode neu programmiert und gestartet werden.

Die Oneshot Interrupt-Latenzzeiten des Experiment-PC sind:

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

  RTAI Testsuite - KERNEL latency (all data in nanoseconds)
  RTH|   lat min|   ovl min|   lat avg|   lat max|   ovl max|   overruns
  RTD|     -2193|     -2193|     -1619|      5757|      5757|   0
  RTD|     -2195|     -2195|     -1604|     10468|     10468|   0
  RTD|     -2195|     -2195|     -1657|      5616|     10468|   0
  RTD|     -2195|     -2195|     -1625|      5730|     10468|   0
  RTD|     -2195|     -2195|     -1658|      5616|     10468|   0
  RTD|     -2195|     -2195|     -1633|      5873|     10468|   0
  RTD|     -2195|     -2195|     -1649|      5629|     10468|   0
  RTD|     -2195|     -2195|     -1629|      5725|     10468|   0
  RTD|     -2195|     -2195|     -1623|      5629|     10468|   0
  RTD|     -2193|     -2195|     -1650|      5623|     10468|   0
```

Abbildung 7.2: oneshot Latenzzeit-Messung im Kernel-Space auf dem Experiment-PC

Es ist zu erkennen, dass die „`ovl max`“ Spalte, satt 5737 ns im periodischen Modus, nun bis zu 10468 ns im „oneshot“ Modus anzeigt. Es treten also im oneshot Modus bis zu doppelt so lange Latenzzeiten auf als im periodischen Modus. Dies würde die maximale Regelfrequenz auf rund 95 kHz absenken.

7.2.1.2 im User-Space

Um die Echtzeitfähigkeit eines Computersystems im User-Space zu bestimmen, können die äquivalenten Testprogramme in Verzeichnis „`/usr/realtime/testsuite/user`“ benutzt werden.

Die Latenzzeiten des Experiment-PC im User-Space sind:

Periodischer Interrupt-Latenzzeiten

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is periodic

 RTAI Testsuite - USER latency (all data in nanoseconds)
 2007/11/3 12:02:17
 RTH|   lat min|   ovl min|   lat avg|   lat max|   ovl max|   overruns
 RTD|     -2912|     -2912|      -275|      3208|      3208|   0
 RTD|     -2854|     -2912|      -275|      2491|      3208|   0
 RTD|     -1339|     -2912|      -275|       864|      3208|   0
 RTD|     -1048|     -2912|      -275|       601|      3208|   0
 RTD|     -1459|     -2912|      -275|       754|      3208|   0
 RTD|     -3261|     -3261|      -275|      2797|      3208|   0
 RTD|     -1218|     -3261|      -275|       805|      3208|   0
 RTD|     -1136|     -3261|      -275|       798|      3208|   0
 RTD|     -5243|     -5243|      -275|      5496|      5496|   0
 RTD|     -5156|     -5243|      -275|      5413|      5496|   0
```

Abbildung 7.3: periodische Latenzzeit-Messung im User-Space auf dem Experiment-PC

Der Vergleich der periodischen Latenzzeiten von Kernel- und User-Space (Abbildung 7.1 und Abbildung 7.3) zeigt keine bedeutenden Abweichungen. Das Latenzzeit-Programm kommuniziert während seiner Ausführung allerdings nicht mit einer Hardware, wie der Mess- und Steuerkarte. Erst der Datenaustausch mit Hardware bewirkt einen Kontextwechsel zu einem Treiberprogramm im Kernel-Space und erhöht damit die Latenzzeit.

Oneshot Interrupt-Latenzzeiten

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

  RTAI Testsuite - USER latency (all data in nanoseconds)
  2007/10/19 11:43:42
  RTH|   lat min|   ovl min|   lat avg|   lat max|   ovl max|   overruns
  RTD|     -1647|     -1647|      1317|      7028|      7028|   0
  RTD|     -1644|     -1647|      1278|     10659|     10659|   0
  RTD|     -1638|     -1647|      1264|      6376|     10659|   0
  RTD|     -1640|     -1647|      1299|      7023|     10659|   0
  RTD|     -1641|     -1647|      1219|     11585|     11585|   0
  RTD|     -1633|     -1647|      1327|      8730|     11585|   0
  RTD|     -1634|     -1647|      1260|      7017|     11585|   0
  RTD|     -1634|     -1647|      1260|      7017|     11585|   0
  RTD|     -1634|     -1647|      1560|      8017|     11585|   0
  RTD|     -1544|     -1647|      1340|      8017|     11585|   0
```

Abbildung 7.4: oneshot Latenzzeit-Messung im User-Space auf dem Experiment-PC

Aus dem Vergleich der oneshot Latenzzeiten von Kernel- und User-Space (Abbildung 7.2 und Abbildung 7.4) ist zu erkennen, dass sich die mittlere Latenzzeit um rund $(1317-(-1607)) = 2924ns$ erhöht hat. Dies hat zur Folge, dass im User-Space Echtzeit-Programme nur mit einer größeren mittleren Periodendauer ausgeführt werden können. Dagegen zeigt die maximal auftretende Latenzzeit („`ovl max`“) eine Erhöhung von rund 1100 ns. Diese Erhöhung ist gering und könnte durch einen einzelnen Ausreißerwert entstanden sein.

7.2.2 Test eines Scilab/Scicos-Programms

Im Hinblick auf die zukünftige Regelfunktion des Experiment-PC, welche mit Scilab/Scicos und Comedi ausgeführt werden soll. Wird in diesem Kapitelpunkt ein Testprogramm mit dem Workflow von Scilab/Scicos entworfen. Dieses Testprogramm soll alle äußeren Funktionen eines Regelprogramms übernehmen und darüber Auskunft geben mit welcher Regelfrequenz ein theoretisch optimaler Regelalgorithmus ausgeführt werden kann. Theoretisch optimal bedeutet, dass der Regelalgorithmus einen Wert über den Analogen Eingang der Mess- und Steuerkarte einliest und direkt über den analogen Ausgang der Mess- und Steuerkarte ausgibt. Durch das Testprogramm werden also keine weiteren Berechnungen vorgenommen. Schneller als dieses Testprogramm kann das zukünftige Regelprogramm nicht laufen.

Um einen Regelalgorithmus für das Experiment umzusetzen, müssen Lichtintensitäten über einen Detektor eingelesen und Regelspannungen an das Piezo-Element angelegt werden. Es wird also mindestens ein analoger Eingang und ein analoger Ausgang benötigt. Das Testprogramm leitet den Eingang sofort an den Ausgang weiter.

Das Scilab/Scicos Testprogramm hat in der Datenfluss-GUI folgenden Aufbau:

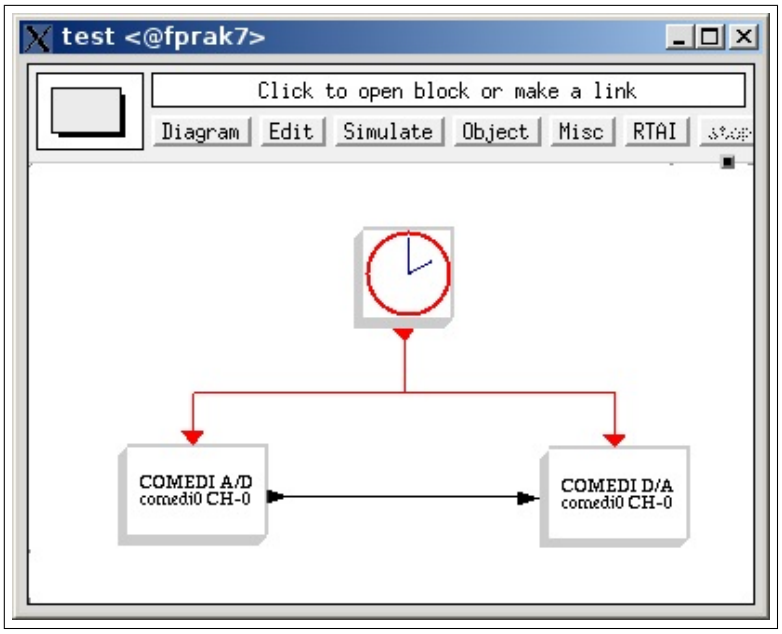

Abbildung 7.5: Testprogramm in Scicos

Dieses Testprogramm kann mit einer festgelegten Periodendauer ausgeführt werden. Bei der Ausführung des Testprogramms wurde ein Frequenzgenerator an den analogen Eingang angeschlossen und ein Oszilloskop an den analogen Ausgang. Dadurch konnte die direkte Umsetzung des Signals durch den Experiment-PC beobachtet werden.
Sollte beim Start von Scicos-Programmen der Fehler „`Segmentation fault`“ erscheinen, so kann es daran liegen, dass vorher nicht alle notwendigen RTAI-Module geladen wurden. Wird das Skript in Abbildung 6.6 fehlerfrei ausgeführt, so sollten alle RTAI-Module bereitstehen.

Auswertung

Mit dem Frequenzgenerator wurde eine 50 Hz Sinus-Frequenz gleichzeitig auf dem Oszilloskop angezeigt und in den analogen Eingang des Experiment-PC eingeleitet. Der Experiment-PC liest das Signal ein und gibt es ohne zusätzliche Verarbeitung auf dem analogen Ausgang aus. Der analoge Ausgang des Experiment-PC wurde ebenfalls, zum Vergleich, auf dem Oszilloskop angezeigt. Das Testprogramm wurde auf dem Experiment-PC mit einer Frequenz von 10 kHz und 100 kHz ausgeführt.

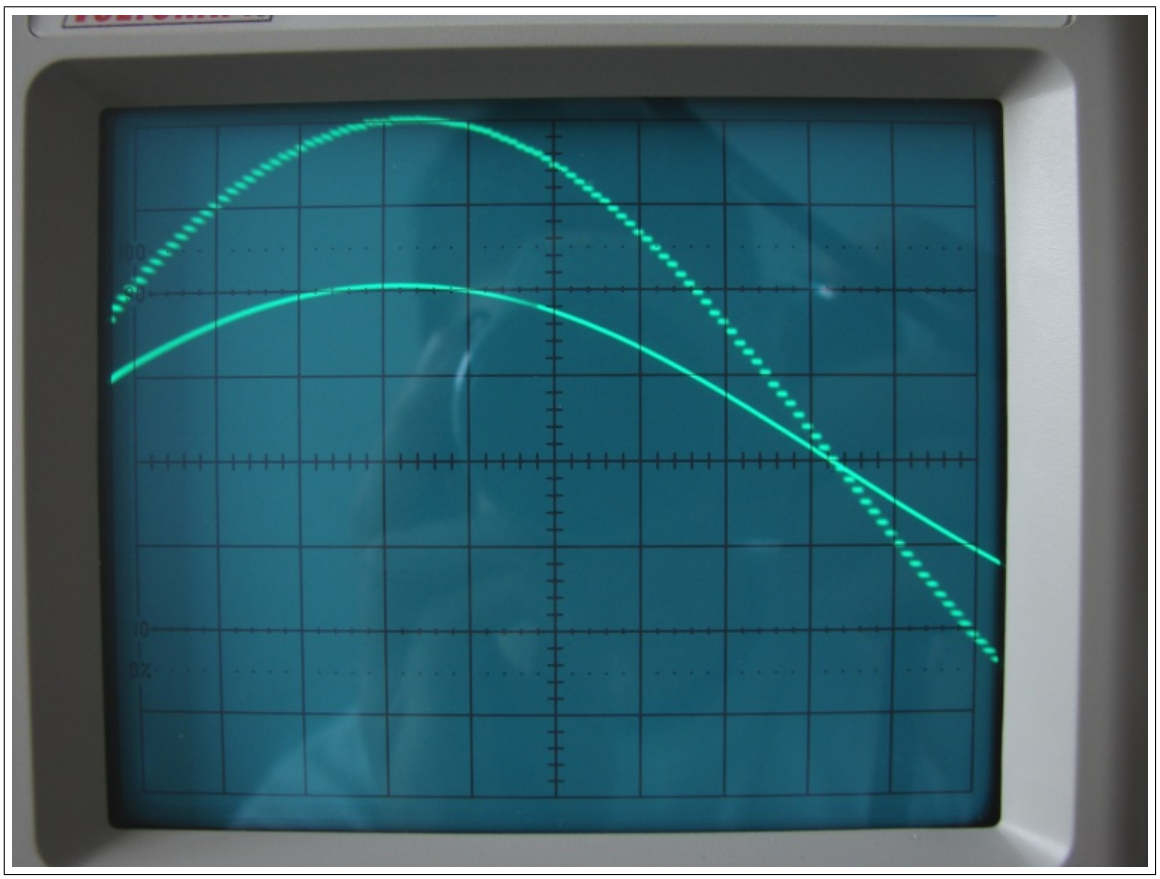

Abbildung 7.6: Testprogramm mit 10 kHz

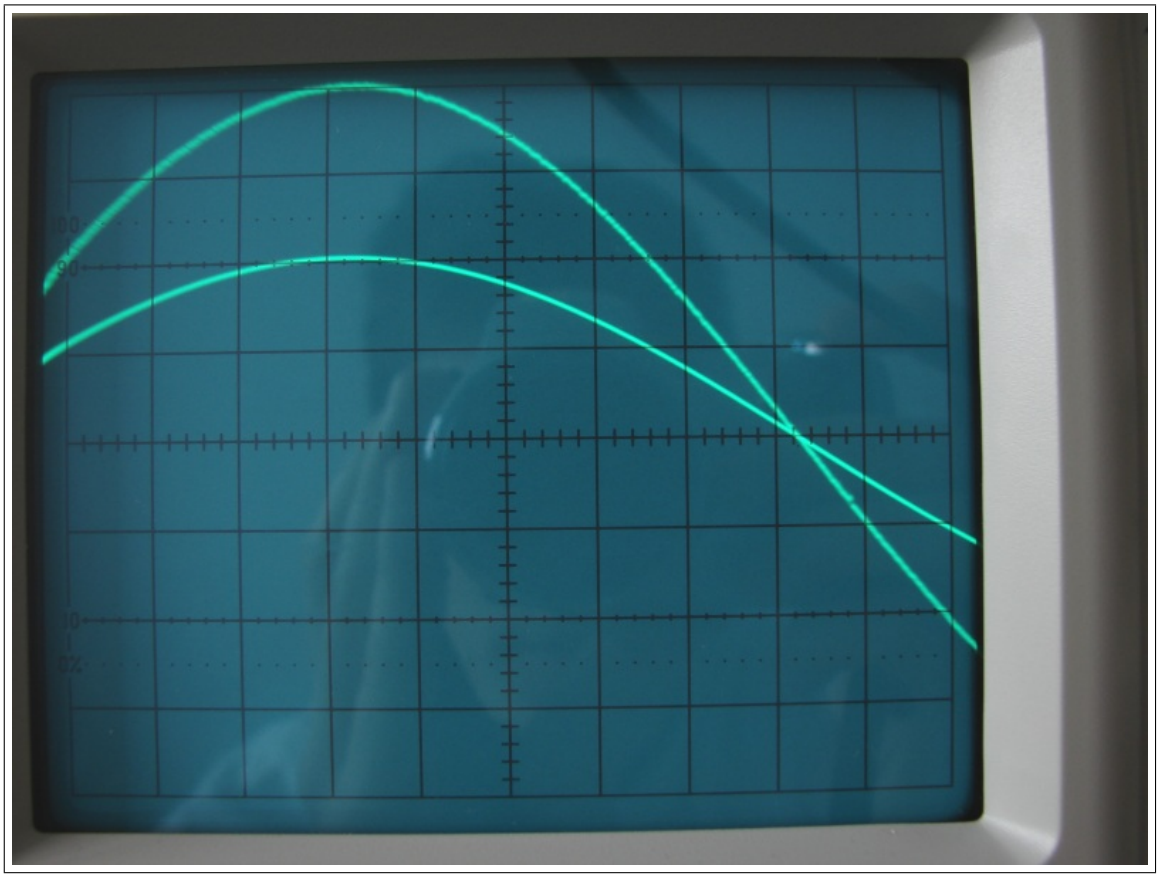

Abbildung 7.7: Testprogramm mit 100 kHz

In Abbildungen 7.6 und 7.7 ist Kanal 1 des Oszilloskops (unteres Signal 0,5 V/div) an das 50 Hz Signal des Frequenzgenerators gekoppelt. Kanal 2 des Oszilloskops (oberes Signal 0,25 V/div) zeigt die Ausgabe des Experiment-PC.
In Abbildung 7.6 ist zu erkennen, dass das Ausgabesignal des Experiment-PC bei einer Frequenz von 10 kHz auf dem Oszilloskop erkennbare Stufen hinterlässt. Jede einzelne Stufe liegt für 0,0001 s an. So ergibt sich eine Einteilung der 50 Hz Eingangsfrequenz in 200 Stufen pro Periode.
Wird die Periodendauer des Testprogramms auf 100 kHz erhöht, so zeigt sich Abbildung 7.7 auf dem Oszilloskop. Es existieren nun 2000 Stufen pro Periode. Eine Zeitdiskretisierung ist in Abbildung 7.7 mit bloßem Auge nicht mehr zu erkennen. Der Nachteil einer Periodendauer von 100 kHz ist, dass der Experiment-PC nicht mehr ansprechbar ist. Um die Echtzeitanforderung des Testprogramms bei 100 kHz zu erfüllen, wird nur noch das Testprogramm ausgeführt. Der Linux-Kernel erhält keine CPU-Zeit mehr, so dass der Experiment-PC einfriert. Das Testprogramm wird jedoch weiterhin stabil ausgeführt.
In der späteren praktischen Anwendung von Scicos-Programmen muss es gewährleistet sein, dass der Experiment-PC nicht einfriert und das Scicos-Programm noch beendet werden kann. Da eine Regelung in Scicos niemals schneller ausgeführt werden kann als das hier behandelte Testprogramm wird die Regelfrequenz des späteren Scicos-Programms unterhalb der 100 kHz liegen. Selbstverständlich ist die exakte Regelfrequenz des späteren Scicos-Programms davon abhängig, wie umfangreich das Regelalgorithmus ist, aber die Größenordnung der Regelfrequenz muss im Bereich von 10^4 Hz liegen, damit der Experiment-PC nicht einfriert.

7.3 Einsatz von LabVIEW und Comedi auf dem Experiment-PC

Da auf dem Experiment-PC die Software LabVIEW installiert und mit Comedi erweitert wurde, soll in diesem Kapitelpunkt ein Test der Regelfähigkeit unter LabVIEW erfolgen. Es wird analog zum Kapitelpunkt 7.2.2 (Test eines Scilab/Scicos-Programms) verfahren und ein „**V**irtuelles **I**nstrument“ (kurz VI) zum Test in LabVIEW erstellt.
Das „Test-VI“ hat, wie das Testprogramm in Scicos, die einzige Aufgabe elektrische Spannungspegel über die Mess- und Steuerkarte einzulesen und sie ohne Verarbeitung über den analogen Ausgang der Mess- und Steuerkarte auszugeben. Mit diesem einfachen Test-VI kann die maximale Verarbeitungsgeschwindigkeit ermittelt werden. Schneller als dieses Test-VI kann eine zukünftige Regelung in LabVIEW nicht erfolgen.

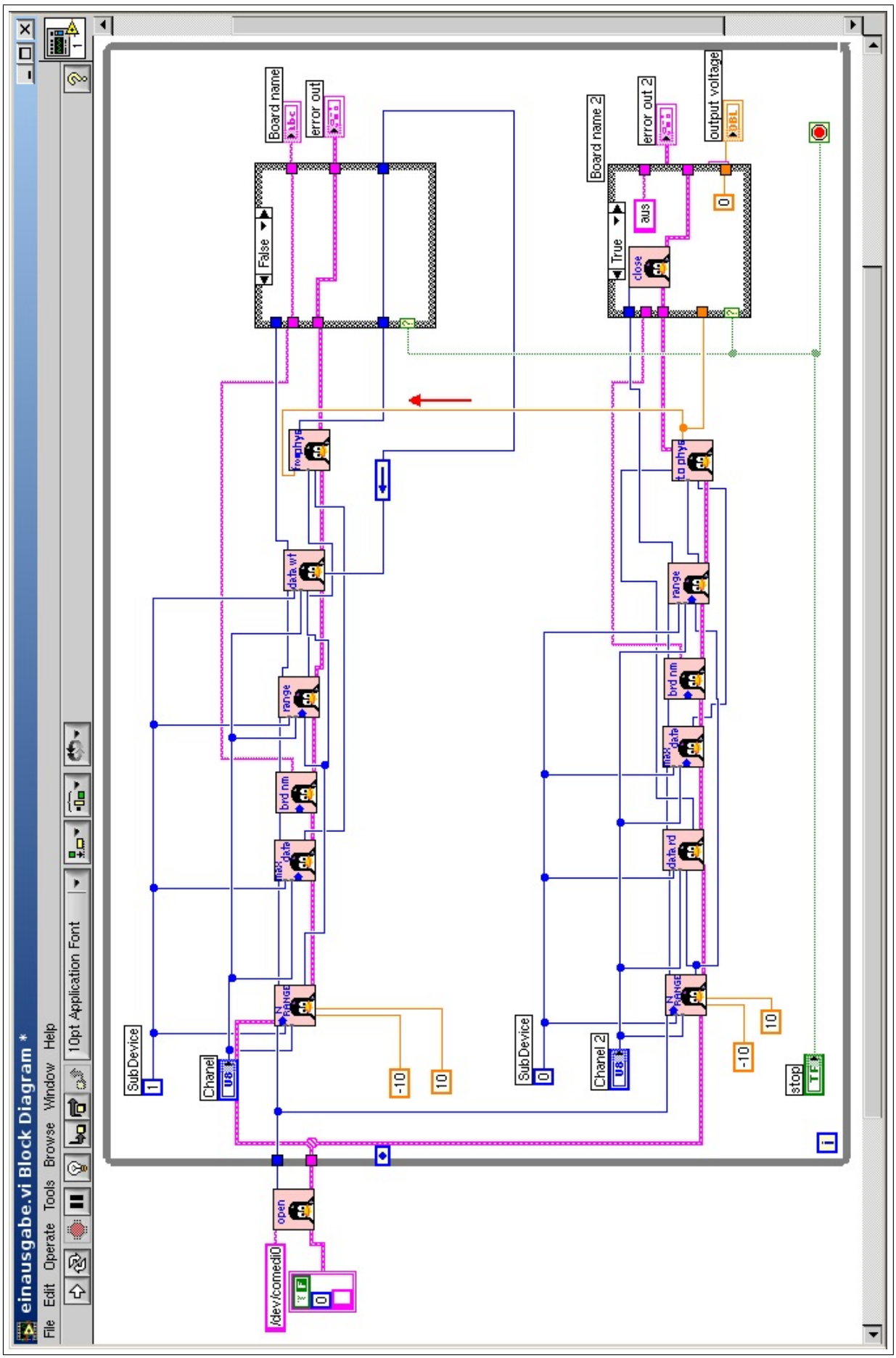

Abbildung 7.8: Test-VI in LabVIEW

In Abbildung 7.8 ist das besagte Test-VI in der Datenfluss-GUI von LabVIEW dargestellt. Die rosafarbenen Sub-Vis, mit den Pinguin-Icons, sind ebenfalls virtuelle Instrumente von LabVIEW. Sie stellen die Kommunikation mit der Mess- und Steuerkarte über die Comedi-API her. Jedes einzelne Comedi-VI enthält eine einzelne Comedi-Funktion. Diese Comedi-Funktionen entsprechen der „User-Space“ Spalte der Tabelle 4.1 und sind keine Funktionen, welche in Echtzeit ausgeführt werden. Die entsprechenden User-Space Comedi-Funktionen werden dann aus der Shared Library „libcomedi.so“ gelesen.
Der rote Pfeil in Abbildung 7.8 zeigt die Übertragungsrichtung des gelesenen Spannungspegels an. Über die, mit dem roten Pfeil, markierte Leitung wird der Spannungspegel aus dem unteren Block, der Spannungspegel mit der Mess- und Steuerkarte einliest, in den oberen Block, der Spannungspegel über die Mess- und Steuerkarte ausgibt, übertragen.
Es sei weiter erwähnt, dass das Test-VI im User-Space von LabVIEW ausgeführt wird. Es existieren in dem Test-VI keine Timer. Das Test-VI wird also vom Experiment-PC so schnell wie möglich ausgeführt.

Auswertung

Hier wurde der selbe Versuchsaufbau wie in Kapitelpunkt 7.2.2 verwendet. Es soll also eine 50 Hz Sinus-Frequenz so schnell wie möglich mit dem Experiment-PC eingelesen und wieder ausgelesen werden. Folgende Abbildung 7.9 zeigt das kombinierte Bild des Oszilloskops:

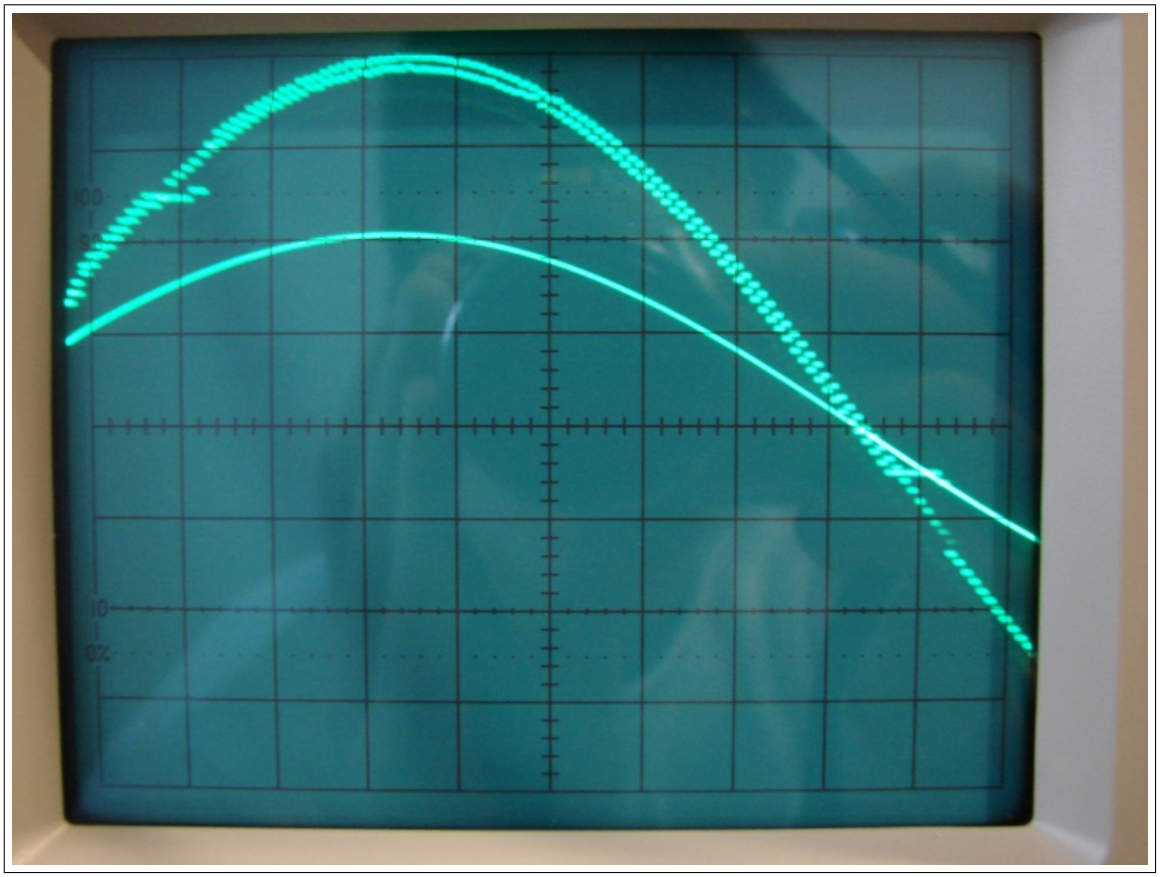

Abbildung 7.9: Test-VI, so schnell wie möglich, auf dem Experiment-PC

In Abbildung 7.9 ist ein analoges Bild zu Abbildung 7.6 oder Abbildung 7.7 zu erkennen, nur mit dem einzigen Unterschied, dass die Ausführung von LabVIEW in nicht-Echtzeit umgesetzt wird. In Abbildung 7.9 sind deutliche Totzeiten von rund 0,4 ms zu erkennen. In diesen Totzeiten bleibt der zuletzt gemessene Wert am analogen Ausgang erhalten, erst nach jeder Totzeit springt der Ausgang auf den aktuellen Spannungspegel. Die Totzeiten erhöhen sich bis auf rund 40 ms, wenn mit dem Experiment-PC gleichzeitig eine andere Tätigkeit ausgeführt wird.
Außerdem sind zusätzliche Zeitverschiebungen aufgetreten, diese machen sich in Abbildung 7.9 durch die zwei sichtbaren Ausgabeverläufe bemerkbar. Da der Experiment-PC das Test-VI mit maximaler Leistung ausführt ist die CPU zu 100% ausgelastet. Bei 100% CPU-Leistung ergibt sich so eine theoretisch maximale Regelfrequenz von (11000 ± 500) Hz. Dieser Wert gilt nur wenn keine Hintergrundaktivität des Experiment-PC vorliegt.
Eine Regelung mit LabVIEW kommt, bei diesem Ergebnis, nur für „langsame“ Störgrößen, wie Temperaturänderungen, in Frage.

Kapitel 8

Ausblick

In diesem Kapitel sollen einige mögliche Verbesserungen und Weiterentwicklungen des Experiment-PC und des Experimentes vorgestellt werden.

8.1 Verbesserungen des RTAI-Gentoo-Linux

Die einzelnen Softwarepakete von Gentoo-Linux sind jederzeit mit dem „emerge“-Tool aktualisierbar. Teilweise können auch mehrere Versionen von Softwarepaketen nebeneinander auf dem Experiment-PC gehalten werden. Um den Portage-Tree auf dem Experiment-PC zu aktualisieren kann der Befehl „`$ emerge --sync`“ benutzt werden. Dadurch werden die ebuilds aller neusten verfügbaren Versionen aus den Experiment-PC geladen. Theoretisch sollten bei zukünftigen Aktualisierungen oder Updates von Softwarepaketen auf dem Experiment-PC keine Fehlfunktionen entstehen, was aber praktisch nicht ausgeschlossen ist.

Eine Möglichkeit das RTAI Gentoo-Linux zu verbessern wäre es den neusten GCC zu verwenden. Zur RTAI-Erweiterung des Experiment-PC wurde nur die GCC Version 3.3.6 verwendet. Ebenfalls kann ständig nach einer neuen Version des RTAI selbst Ausschau gehalten werden. Ein Update von RTAI selbst wäre lohnenswert, in der Hinsicht auf die Beseitigung von gefundenen Bugs oder die Verringerung der Latenzzeiten des Experiment-PC. Um ein Update der RTAI Version durchzuführen, müssten die Gliederungspunkte 2.4 bis 2.13 des [rtaitut] mit der neuen RTAI Version durchgeführt werden. Eine andere Möglichkeit wäre es, neue Linux-Kernel-Versionen auszuprobieren in der Hoffnung bessere Latenzzeiten oder einen stabileren Experiment-PC zu erhalten. In dieser Arbeit wurde RTAI mit den Linux-Kernel-Versionen 2.6.21, 2.6.19.7 und 2.6.19 mit den selben Latenzzeiten und Stabilitäten getestet.

8.2 Fortsetzungen für das Experiment

Der Experiment-PC steht nun bereit eine Regelung in Echtzeit zu realisieren. Die Regelung ist ein herkömmliches Echtzeit-Programm, welches die Eingabesignale aus der Mess- und Steuerkarte verarbeitet und das geeignete Ausgabesignal über die Mess- und Steuerkarte ausgibt. Ein entwickelter Regelalgorithmus kann gleichermaßen für die zwei Teilexperimente aus Kapitel 2 verwendet werden. Es stehen nun mehrere Wege zur Verfügung, um eine Echtzeit-Regelung zu erstellen.

Diese sind:

- Erstellung des Echtzeit-Programms mit Scilab/Scicos, sowie dessen Manipulation über RTAI-Lab.
- Programmierung eines Scilab/Scicos unabhängigen Kernel-Moduls mit dem GCC. Dieses Kernel-Modul beinhaltet die Echtzeit-Regelung und kann nur durch den Administrator des Experiment-PC geladen werden. Bei geeigneter Kombination von FIFIO kann dieses Kernel-Modul über RTAI-Lab manipuliert werden.
- Programmierung eines Scilab/Scicos unabhängigen User-Space-Programms mit dem GCC. Damit das User-Space-Programm in Echtzeit ausgeführt werden kann, muss LXRT verwendet werden. Das User-Space-Programm kann von einem normalen Benutzer gestartet werden und benötigt keine Administratorrechte. Der Nachteil ist, dass durch LXRT, das heißt ein Echtzeit-Programm im User-Space, eine längere Latenzzeit entsteht und das User-Space-Programm nur mit einer kleineren maximalen Regelperiodenfrequenz ausgeführt werden kann.

8.2.1 Regelung in Hardware

Es gibt eine weitere interessante Möglichkeit die benötigte Regelung für das Experiment umzusetzen. Diese Möglichkeit benötigt keine Echtzeitfähigkeit des Experiment-PC und wird nur indirekt durch den Experiment-PC ausgeführt.
Die Mess- und Steuerkarte ME-4660i verfügt über den FPGA „Spartan II XC2S150“, in Abbildung 2.4 rot markiert. Dieser FPGA ist ein integrierter Mikrochip, der rekonfigurierbar ist. Das bedeutet, der Benutzer kann eine beliebige logische Schaltung in Hardware beliebig oft und verschieden auf den FPGA konfigurieren. Da der FPGA das „Herzstück“ der ME-4660i ist, hat er Zugriff auf Speicher, Register und alle Analog/Digitalwandler der Mess- und Steuerkarte.
Momentan wird der FPGA der ME-4660i nur mit der Aufgabe konfiguriert, alle durch den A/D oder D/A-Wandler ermittelten Daten an den PCI-Bus des Experiment-PC weiterzuleiten. Dies ist die Firmware der Mess- und Steuerkarte, siehe Kapitelpunkt 6.4.6. Dadurch kann das RTAI Gentoo-Linux Daten, über Register, ein- oder auslesen.

Die Konfiguration des FPGAs der ME-4660i könnte nun so abgeändert werden, dass der FPGA selbstständig eine Regelung ausführt. Diese Regelung würde dann vollständig in Hardware realisiert.

Ein großer Vorteil der Regelung in Hardware wäre die enorme Zuverlässigkeit und Geschwindigkeit. Ist der Konfigurations-Bitstream einmal durch den Experiment-PC auf den FPGA geschrieben worden, so ist die Regelung vom Experiment-PC unabhängig und wird nur durch den FPGA mit 33 MHz ausgeführt. Die logischen Schaltungen von FPGAs sind sehr zuverlässig und können nicht durch andere „Tasks“ unterbrochen oder gestört werden, wie das im RTAI Gentoo-Linux möglich ist.

Workflow

Die Fähigkeiten des FPGA werden durch einen Bitstream bestimmt, der in der Praxis in einer Datei abgespeichert wird. Die Firmware der ME-4660i, welche bei jedem Start des Experiment-PC auf die ME-4660i geladen werden muss, ist ein solcher Bitstream. Die Firmware konfiguriert den FPGA allerdings nur mit den Fähigkeiten der Weiterleitung von Daten.
Mit dem Experiment-PC könnte unter Benutzung von Software-Tools der Firma Xilinx ein neuer Bitstream „synthetisiert“ werden, der die Regelung in Hardware umsetzt. Dieser neue Regel-Bitstream könnte auf analogem Wege, wie die Firmware, auf die ME-4660i geladen werden und den FPGA konfigurieren. Der FPGA würde dann sofort nach der Konfiguration die logische Schaltung in Betrieb nehmen.

Literaturverzeichnis

[ADwin] Homepage der Jäger Computergesteuerte Messtechnik GmbH, http://www.adwin.de/de/start/echtzeit.html

[fabres] Experiment 03 - FABRY PEROT RESONATOR, enp03.pdf

[gentoohb] Gentoo Linux x86 Handbuch, http://www.gentoo.org/doc/de/handbook/handbook-x86.xml

[hbtechreg] Winfried Oppelt: Kleines Handbuch technischer Regelvorgänge, Weinheim 1954

[mikauto] Dieter Kanton: Mikroprozessorsysteme in der Automatisierungstechnik, Berlin 1978

[nvidiaguide] Gentoo Linux nVidia Guide, http://www.gentoo.org/doc/de/nvidia-guide.xml

[rtaitut] RTAI-Lab tutorial: Scilab, Comedi, and real-time control, https://www.rtai.org/RTAILAB/RTAI-Lab-tutorial.pdf

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

ABS	**A**nti**b**lockier**s**ystem
API	**A**pplication **P**rogramming **I**nterface
D-Regler	**D**ifferential-Regler
ES	**E**ingebettetes **S**ystem
eCos	**e**mbeded **C**onfiguration **o**perating **s**ystem
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FIFO	**F**irst-**I**n-**F**irst-**O**ut
GCC	**G**NU-**C**-**C**ompiler
GUI	**G**raphical **U**ser **I**nterface
HL-L	**H**alb**l**eiter **L**aser
HAL	**H**ardware **A**bstraction **L**ayer
He-Ne-L	**He**lium-**Ne**on **L**aser
I/O	**I**n-/**O**utput
I-Regler	**I**ntegral-Regler
MBX	**M**ail**b**o**x**
MUP	**M**ulti-**U**ni-**P**orzessor
PC	**P**ersonal **C**omputer
PID	**P**roportional-**D**ifferential-**I**ntegral-Regelung
P-Regler	**P**roportional-Regler
RT	**R**eal **T**ime
RTAI	**R**eal **T**ime **A**pplication **I**nterface
RTHAL	**R**eal **T**ime **H**ardware **A**bstraction **L**ayer
SEM	**Sem**aphore
SMP	**S**ymetric **M**ulti**p**rocessing
VI	**V**irtuelles **I**nstrument

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, den 04.02.2008 Markus Jäger