Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Joachim Hagedorn

Entwicklung und Realisierung eines Echtzeitsystems zur Durchführung von AD/DA-Umsetzungen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Joachim Hagedorn

Entwicklung und Realisierung eines Echtzeitsystems zur Durchführung von AD/DA-Umsetzungen

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.Ing. Florian Wenck
Zweitgutachter : Prof. Dr. Robert Heß

Abgegeben am 14. Januar 2016

Joachim Hagedorn

Thema der Bachelorthesis

Entwicklung und Realisierung eines Echtzeitsystems zur Durchführung von AD/DA-Umsetzungen

Stichworte

QNX, Echtzeitsystem, Echtzeitbetriebssystem, ADC, DAC, Interrupt, Scheduling

Kurzzusammenfassung

Die Arbeit umfasst ein Echtzeitsystem mit QNX als Echtzeitbetriebssystem, das analoge Signale in digitale Werte umwandelt, um im Anschluss diese digitalen Werte erneut in ein analoges Signal zu überführen und auszugeben. Dieses System wird mit unterschiedlichen Echtzeit-Scheduling-Verfahren durchgeführt und auf dessen Echtzeitfähigkeit getestet.

Joachim Hagedorn

Title of the paper

Development and implementation of a real-time system for performing AD/DA-conversions

Keywords

QNX, real-time system, RTOS, ADC, DAC, interrupt, scheduling

Abstract

The report includes a real-time system, which uses QNX as the real-time operating system. This system converts analog signals into digital values, in order to convert these digital values again to an analog signal for putting it out. This system is carried out with different real-time scheduling methods and is tested for its real-time capability.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

In der heutigen Zeit bekommen Echtzeitsysteme einen immer größeren Stellenwert in der Industrie. Die Sicherheit des menschlichen Lebens ist von großer Bedeutung. Echtzeitsysteme kommen in sicherheitsrelevanten Systemen vor. Daher ist Echtzeit in der Prozessautomatisierung ein wichtiges Thema. Automobil-Hersteller entwickeln Echtzeitsysteme in Antiblockiersystemen. Darüber hinaus kommen diese Systeme in Weltraum-Systemen vor, wie zum Beispiel in Satelliten und Weltraumstationen. Ein weiteres sicherheitsrelevantes Gebiet, in der Echtzeitsysteme erforderlich sind, ist die Energiebranche. Im speziellen werde diese für Energieverteilungssystemen verwendet, vgl. [7, Seite 1]. Überall dort, wo programmierbare elektronische Systeme eingesetzt werden, die Leib und Leben in Gefahr bringen, werden Echtzeitsysteme benötigt. Diese stellen sicher, dass bestimmte Abläufe zu einer ganz bestimmten Zeit ausgeführt werden.

1.1 Motivation

Das Thema der Echtzeitsysteme und dessen Programmierung ist Teil des Masterstudiengangs “Automatisierung“ in dem Kurs “Betriebssysteme und Echtzeitprogrammierung“. In diesem Themengebiet der Echtzeitsysteme gibt es bislang keine praktische Aufgabe im Labor, um das Verständnis in diesem Gebiet zu festigen und auszubauen. Die Studierenden haben somit nicht die Möglichkeit, anhand eines praktischen Beispiels die wichtigen Aspekte dieser Systeme zu erfahren. Daher wird ein Echtzeitsystem für das Labor als Aufgabe entwickelt, welches den Studierenden den Umgang mit der Echtzeitprogrammierung näher bringt.

1.2 Ziele der Arbeit

Die Arbeit beschäftigt sich mit einem System, welches in Echtzeit ausgeführt wird. Dieses System besteht zum einen aus einem ADC, dessen Aufgabe es ist, analoge Signale abzutasten und als digitalen Wert im System abzuspeichern und zum anderen aus einem DAC,

der die digitalen Werte in analoge Werte umwandelt und diese ausgibt. Die beiden Prozesse stellen die Kernaufgabe des Systems dar, welche möglichst sicher durchgeführt werden sollen. In der Echtzeitprogrammierung spielt der Scheduler eines Betriebssystems eine sehr wichtige Rolle. Dieser organisiert die zeitliche Nutzung der Prozesse, die der Prozessor als Ressource zur Verfügung stellt. Bei QNX, welches das Betriebssystem des Systems ist, gibt es drei verschiedene Scheduling-Verfahren. Diese haben unterschiedliche Vorgehensweisen und Qualitäten. Um diese Qualitäten der Scheduling-Verfahren festzustellen, die sich für die ADC- / DAC- Aufgabe am besten eignen, wird der Kern des Systems bei jedem Scheduling mittels Belastung geprüft. Die Belastung besteht dabei aus einer Reihe von Counter-Threads, die lediglich eine Variable hochzählen, um unnötige Prozessorzeit in Anspruch zu nehmen. Bei einer gewissen Anzahl an Counter-Threads bricht das gesamte System zusammen. Diese Anzahl ist die Qualitätsaussage der benutzten Scheduling-Verfahren.

1.3 Gliederung der Arbeit

Die Arbeit ist in acht Kapitel aufgeteilt. In dem ersten Kapitel “Einleitung“ wird das Thema vorgestellt. Es beinhaltet die Einführung in das Thema der Echtzeit, die Ziele der Arbeit und die Beschreibung der Gliederung der Arbeit. Das nächste Kapitel beschäftigt sich mit der Erläuterung der Grundlagen dieser Arbeit. Dabei wird der Begriff der Echtzeit und dessen Bedingungen, die Funktionsweise des Betriebssystems QNX, Prozesse und Threads, Scheduling, ADC / DAC, Interrupts und Polling erklärt. Die für die Arbeit benötigten Komponenten werden im darauffolgenden Kapitel beschrieben. Diese sind unterteilt in Software und Hardware. Die Hardware besteht aus dem eingebetteten System, Oszilloskop, Funktionsgenerator und der Steuereinheit, die digitale Eingangswerte erzeugt. Die Software besteht aus dem Betriebssystem QNX Neutrino, der Entwicklungsumgebung QNX Momentics IDE, dem Programm “Multi Channel Software“ für das Oszilloskop und dem Treiber “Universal Driver“ von Diamond Systems. Das nächste Kapitel beinhaltet das Konzept und das Design der Arbeit. Dabei wird die Herangehensweise und die Funktionalität der Arbeit beschrieben. Weiterhin ist die Arbeit und dessen Funktionsweise anhand eines Aktivitätsdiagramms grafisch dargestellt. Die Realisierung der Arbeit wird im folgenden Kapitel beschrieben. Dies ist die Beschreibung des Programmcodes für das zu entwickelnde, echtzeitfähige Programm. Im Anschluss folgt das Kapitel “Qualitätsprüfung der Scheduling-Verfahren“. Dieses beinhaltet das Test-Verfahren, welches durch ein Shell-Skript realisiert wird. Da die Arbeit für den Kurs “Betriebssysteme und Echtzeitprogrammierung“ entwickelt wird, stellt das nächsten Kapitel eine Versuchsbeschreibung für das Labor dar. Das letzte Kapitel befasst sich mit einer Zusammenfassung dieser Arbeit. Gleichwohl wird ein Ausblick gewährt. Dieser gibt Ansätze zur weiteren Bearbeitung dieser Arbeit.

2 Grundlagen

Das Grundlagenkapitel beschäftigt sich mit Erläuterungen gängiger Grundbegriffe, die im Zusammenhang mit dieser Arbeit stehen. Unter anderen wird das Prinzip der Echtzeit beschrieben, welche Bedeutung diese hat, und wo diese zum Einsatz kommt. Des Weiteren wird der System-Kern der Arbeit erläutert. Dieser Kern besteht aus dem ADC und aus dem DAC. Darin enthalten ist die Funktionsweise der beiden Systeme. Auf dem eingebetteten System “Hercules II“ von Diamond Systems ist das Betriebssystem QNX installiert. Dieses Betriebssystem und dessen Vor- und Nachteile sind ebenfalls Bestandteil des Grundlagenkapitels. Weiterhin werden allgemeine Begriffe aus dem Bereich der Betriebssysteme, wie Interrupts, Polling und Scheduling erläutert.

2.1 Echtzeit

Im Allgemeinen bedeutet Echtzeit, dass ein System, neben der Korrektheit der Abarbeitung, auf periodische und aperiodische Ereignisse innerhalb eines gewissen Zeitraums in allen Betriebssituationen reagiert [7, Seite 1 ff][8]. Es wird dabei auch von logischer und zeitlicher Korrektheit gesprochen [1, Seite 317]. Ereignisse, die periodisch auftreten, sind einfacher zeitlich vorhersehbar, weil sie zu jedem Zeitpunkt die gleiche Abarbeitungsdauer in Anspruch nehmen. Dagegen können zusätzlich asynchron auftretende Ereignisse den zur Verfügung stehenden Zeitrahmen überschreiten. Ist ein System echtzeitfähig, so wird der Zeitrahmen trotz aperiodischer Ereignisse nicht überschritten. Dies hat zur Folge, dass alle benötigten Daten zu den entsprechenden Zeiten sicher zur Verfügung stehen.

Die zeitliche Korrektheit wird mit dem Beispiel aus Abbildung 2.1 verdeutlicht. Zu sehen sind zwei periodische Ereignisse (A, B), die in Abstand T auftreten und ein aperiodisches Ereignis (C), das kurz vor der dritten Periode einmal auftritt. Der Zeitrahmen zwischen den Perioden ist so gewählt, dass die Gesamtbearbeitungszeit der Ereignisse im schlimmsten Fall nicht größer ist als eine Periode. Das bedeutet, dass unter normalem Betrieb stets ein Zeitpuffer (P) vorhanden ist, der nicht genutzt wird. Das zeigt, dass Echtzeitsysteme nicht effektiv, sondern sicher arbeiten müssen. Tritt ein asynchrones Ereignis kurz vor der Abarbeitung des periodischen Ereignisses auf, wird erst dieses und im Anschluss die periodischen Ereignisse

abgearbeitet. Somit entsteht eine längere Bearbeitungszeit. Dennoch wird die Periode nicht durch die Gesamtbearbeitungszeit überschritten.

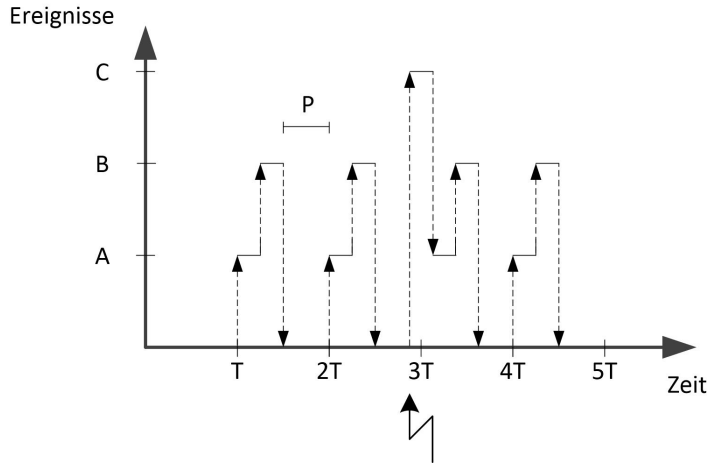

Abbildung 2.1: Ablauf einer Echtzeitabarbeitung

Die Überschreitung von zeitlichen Grenzen wird je nach Anforderung des Systems unterschieden in:

- **Harte Echtzeitbedingung**

  Unterliegt ein System dieser Bedingung, so müssen die Zeiten unter allen Umständen unbedingt eingehalten werden. Dies ist erforderlich, wenn das Nichteinhalten von Zeiten menschliches Leben gefährdet oder große Schäden verursacht.

- **Feste Echtzeitbedingung**

  Bei der festen Echtzeitbedingung werden die Informationen nach dem Überschreiten der zeitlichen Grenzen wertlos. Jedoch resultiert daraus kein Schaden. Die verspäteten Informationen sind zwar unbrauchbar, dennoch kann das System seine Arbeit fortsetzen, ignoriert diese Informationen und wiederholt gegebenenfalls die Aktion.

- **Weiche Echtzeitbedingung**

  In dieser Variante dürfen die Zeitbedingungen überschritten werden. Dabei kann eine Toleranzgrenze festgelegt werden, in der verspätete oder zu früh auftretende Ereignisse auftreten dürfen. Dies ist der Fall, wenn diese Überschreitungen häufig vorkommen. Des Weiteren kann ein Ereignis die Grenzen auch weit überschreiten, wenn die Nichteinhaltung selten passiert. Die Informationen behalten in beiden Situationen ihre Wertigkeit.

Wie oben beschrieben, unterliegen Echtzeitsysteme zeitlichen Anforderungen. Diese werden im Speziellen durch Rechtzeitigkeit, Gleichzeitigkeit und Verfügbarkeit definiert.

2.1.1 Rechtzeitigkeit

Die Rechtzeitigkeit beschreibt im Allgemeinen, dass Ereignisse zu bestimmten Zeiten auftreten müssen, um somit die daraus gewonnenen Informationen zur Verfügung zu stellen [1, Seite 318 ff]. Es gibt dabei verschiedene Möglichkeiten zur Beschreibung der bestimmten Zeiten. Diese sind abhängig von der Bedingung des Systems.

Zum einen kann dabei der exakte Zeitpunkt t festgelegt sein, in dem das Ereignis stattfinden muss. Dies ist in Abbildung 2.2 a) zu sehen. Dabei darf das Ereignis nicht vor und nicht nach dem Zeitpunkt t auftreten. Zum anderen definiert die Rechtzeitigkeit die Variante, dass das Ereignis spätestens zu einem Zeitpunkt t_{max} aufgetreten sein muss, wie in Abbildung 2.2 b) der gestrichelte Pfeil deutlich macht. Ebenso gilt das für den Fall in Abbildung 2.2 c), nur dass das Ereignis erst ab den Zeitpunkt t_{min} auftreten darf. Als letztes gibt es die Variante, dass innerhalb eines Intervalls zwischen t_{min} und t_{max} das Ereignis auftreten darf. Dies ist in Abbildung 2.2 d) zu sehen.

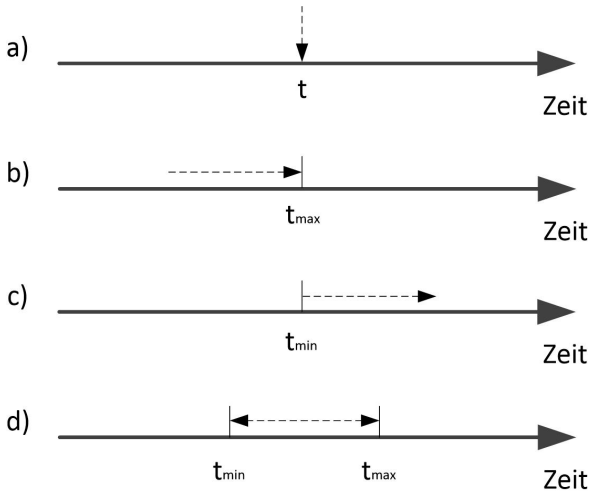

Abbildung 2.2: Varianten der Zeitbedingungen

Zeitbedingungen werden unterschieden in absolute Zeitbedingungen, die eine genaue Uhrzeit vorgeben, wann ein Ereignis ausgelöst werden soll und in relative Zeitbedingungen, die zum Beispiel Bezug auf ein vorheriges Ereignis nehmen. Dabei tritt das nächste 5 Sekunden später auf.

2.1.2 Gleichzeitigkeit

Ein Echtzeitsystem muss im Allgemeinen gleichzeitig auf mehrere Ereignisse reagieren und diese abarbeiten können. [1, Seite 322 ff] Die Schwierigkeit ist dabei die Einhaltung der Rechtzeitigkeit. Zur Bewältigung der Gleichzeitigkeit gibt es unterschiedliche Szenarien:

- Vollständige Parallelverarbeitung in einem Mehrprozessorsystem
- Quasi-parallele Verarbeitung in einem Mehrprozessorsystem
- Quasi-parallele Verarbeitung in einem Einprozessorsystem

Im Fall der vollständigen Parallelverarbeitung wird jedem Prozessor im Mehrprozessorsystem jeweils maximal eine Aufgabe zugeteilt. So können die Aufgaben unabhängig voneinander betrachtet werden, und jede Aufgabe erhält die komplette Verarbeitungsleitung des Prozessors.
Im zweiten Fall verläuft die Verarbeitung quasi-parallel, wenn mehr Aufgaben als Prozessoren vorhanden sind. Dies hat zur Folge, dass sich mehrere Aufgaben die Verarbeitungsleistung eines Prozessors teilen müssen. Um die Zeitverteilung zu steuern, welche die jeweiligen Aufgaben zur Abarbeitung benötigen, muss hierfür ein Echtzeitscheduling durchgeführt werden, um die Zeitbedingungen der jeweiligen Aufgaben einhalten zu können. In einem der folgenden Unterpunkte dieses Kapitels wird auf Scheduling ausführlicher eingegangen.
Der letzte Fall beschreibt die komplette Verarbeitung aller Aufgaben auf einem Prozessor. Dabei wird die Zeitverteilung, wie auch im zweiten Fall, durch einen Echtzeitscheduler verwaltet.

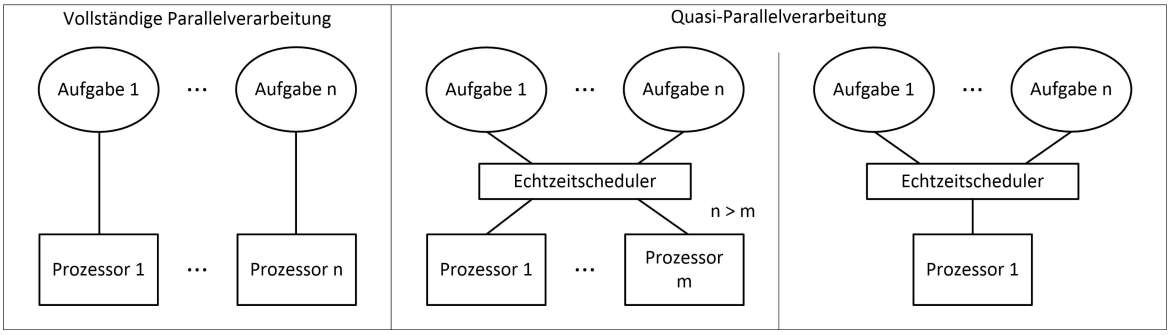

Abbildung 2.3: Varianten der Gleichzeitigkeit [1, Seite 324]

2.1.3 Verfügbarkeit

Echtzeitsysteme müssen zu jeder Zeit verfügbar sein, um die Zeitbedingungen der Aufgaben einhalten zu können und die Rechtzeitigkeit somit gewährleisten zu können. Diese Systeme leisten ihre Arbeit im Allgemeinen über einen langen Zeitraum oder rund um die Uhr. Daher ist es wichtig, dass sie stets zuverlässig ihre Aufgaben abarbeiten. Potenziell kritische Unterbrechungen treten vor allem im Bereich der Reorganisation auf. So kann es sein, dass der Datenbankspeicher während des laufenden Betriebes reorganisiert oder bereinigt wird. Diese Bereinigung wird unter der Programmiersprache Java durchgeführt, die auch als Garbage Collection bekannt ist [1, Seite 325].
Es gibt Möglichkeiten, die Probleme der Unterbrechungen zu reduzieren. Ist die Speicherreorganisation von Datenbanken notwendig, so können stattdessen reorganisationsfreie Algorithmen verwendet werden. Für die Speicherbereinigung gibt es ebenfalls alternative Verfahren, wie in Abbildung 2.4 zu sehen ist. Normalerweise wird die Bereinigung in einem Block durchgeführt. Wird der Block nun in viele kleine Schritte aufgeteilt, so können die Zeitbedingungen dennoch eingehalten werden.

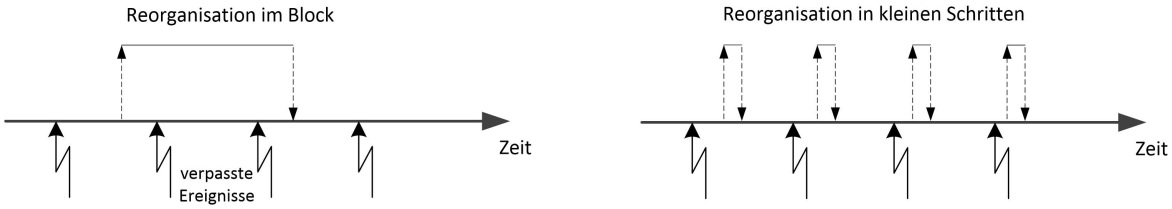

Abbildung 2.4: Varianten für Reorganisationsverfahren [1, Seite 325]

2.2 QNX

QNX ist ein Echtzeitbetriebssystem. Im Vergleich zu normalen Betriebssystemen, die einen monolithischen Kernel besitzen, baut QNX auf einen Mikrokernel auf (Abbildung 2.5). Dessen Besonderheit ist, dass darin lediglich Basisfunktionen, wie die Speicherverwaltung, die Kommunikation und Synchronisation realisiert sind [2]. Alle weiteren Dienste, wie sie vom monolithischen Kernel bekannt sind, werden als Systemtasks ausgeführt. Somit sind beispielsweise die Treiber und das Dateisystem außerhalb des Kernels implementiert. Dies ermöglicht eine flexible Konfiguration des Betriebssystems.

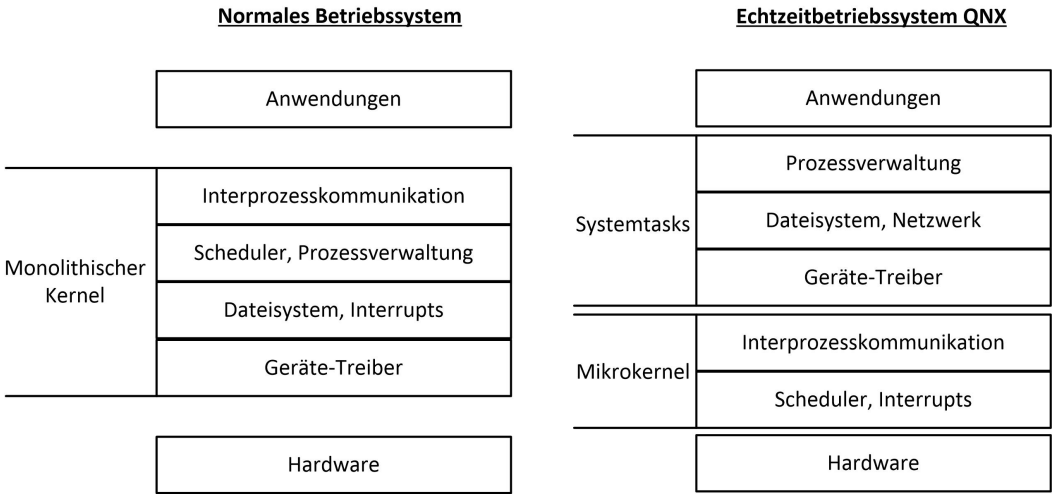

Abbildung 2.5: Vergleich vom normalen Betriebssystem zu QNX [2]

Für eine Installation auf ein eingebettetes System ist ein Betriebssystem mit geringem Ressourcenbedarf erforderlich. Dafür eignet sich ein dediziertes QNX-System. Das bedeutet, dass irrelevante Systemtasks, die zur Bewältigung der Aufgabe unwichtig sind, nicht installiert werden. Es ist somit möglich, das QNX-System auf das Wesentliche zu reduzieren. Bei der Installation aller zur Verfügung stehenden Systemtasks wird QNX als allgemeines Echtzeitbetriebssystem ausgeführt. In dieser Form bietet es die gleiche Funktionalität wie normale Betriebssysteme, mit dem Unterschied, dass die Funktionen auf Echtzeit ausgelegt sind, vgl. [1, 425 ff] [2, 9, 10].

Die elementaren Aufgaben des Mikrokernels von QNX werden im Folgenden aufgelistet.

- **Interprozesskommunikation (IPC)**
  Im Allgemeinen bedeutet Interprozesskommunikation Austausch von Daten zwischen Prozessen oder Threads innerhalb eines Systems. Die Art der Übertragung muss jedoch synchronisiert werden, so dass die Daten erst gesendet und im Anschluss empfangen werden. Die Übertragung wird mit Pipes realisiert. Beide Prozesse oder Threads besitzen jeweils zwei Datenströme. Die eine dient zum Empfangen und die andere zum Senden der Daten. Als gemeinsamer Speicher wird ein FIFO-Puffer (First In First Out) benutzt. Eine weitere Möglichkeit zur IPC ist das Message Queuing. Diese Variante ist ein Anwendungsprotokoll von Microsoft und speichert die Nachrichten in einer Warteschlange. Die Vorgehensweise ist dabei ähnlich wie die der Pipes. Der Unterschied ist, dass die Nachrichten als Grundeinheit in den FIFO-Speicher gespeichert werden und nicht in mehreren Bytes, vgl. [11].
- **Interrupts**
  Interrupts sind Unterbrechungsanforderungen, die von der Hardware generiert wird.

Der Mikrokernel registriert diese Unterbrechung und leitet diese an die dafür vorgesehenen Systemtasks und Anwendungen weiter. Des Weiteren gibt es Software-Interrupts, die auch Systemcalls oder Exceptions genannt werden. Diese Unterbrechungen werden entweder durch das Betriebssystem oder durch eine Software hervorgerufen.

- **Scheduler**
  Zur Verteilung der Prozessorzeit für die Prozesse benutzt der Mikrokernel einen Scheduler. QNX bietet verschiedene Scheduling-Verfahren wie FIFO, Round Robin und Sporadic zum Erreichen einer möglichst effektiven Nutzung des Prozessors und der Einhaltung der Echtzeitfähigkeit. Eine detailliertere Beschreibung des Schedulers steht in eines der folgenden Unterkapitel.

Die Systemtasks einer typischen QNX-Konfiguration setzen sich wie folgt zusammen:

- **Prozessverwaltung**
  Die Prozessverwaltung dient zur Kontrolle der Prozesse. Sie hat die Fähigkeit, Prozesse zu beenden, anzuhalten und Prozessprioritäten festzulegen, um die Ablaufreihenfolge entsprechend der Vorgabe des Schedulers zu verändern.
- **Geräte-Treiber**
  Die Schnittstelle zwischen der Hardware und dem Betriebssystem wird durch den Geräte-Treiber koordiniert. Dabei werden Geräte initialisiert und die Kommunikation zwischen der Software und der Hardware ermöglicht.
- **Dateisystem**
  Die Verwaltung des Dateisystems wird durch diesen Systemtask durchgeführt. Er verwaltet den Speicher und somit die Verzeichnisse, Dateien und FIFOs der Festplatten im System.

Zur angepassten Installation von QNX können, wie bereits beschrieben, Systemtasks hinzugefügt werden oder wegfallen, um die Bedingungen an das System einzuhalten. Zum Beispiel benötigt ein System ohne Festplatte keinen Dateisystemtask.

2.3 Prozesse und Threads

Ein Prozess übernimmt die Aufgabe, mit Hilfe des Prozessors, ein Programm abzuarbeiten. Das Betriebssystem erstellt sozusagen ein Objekt des Programms und dirigiert den Ablauf. So ist das Programm lediglich der Ablaufplan und der Prozess der Durchführende. Das bedeutet, dass ein Programm mehrfach auf einem System ausführbar ist und ein Prozess nur einmal im System vorhanden ist. Alle Prozesse besitzen einen eigenen Speicherbereich,

so dass es nicht ohne weiteres möglich ist, dass ein Prozess Informationen eines anderen Prozesses manipulieren kann. Dies ist nur mit Hilfe der Interprozesskommunikation möglich. Beim Ablauf eines Programms durchläuft ein Prozess verschiedene Zustände, wie in Abbildung 2.6 zu sehen ist. Beim Start des Programms wird ein Prozess erzeugt, der in den Zustand “neu“ übergeht. Ist die Initialisierung abgeschlossen, meldet sich dieser bei dem Scheduler als “bereit“. Das bedeutet, dass der Prozess noch keinem Prozessor zugewiesen ist und demnach nicht läuft, aber zum Ablauf bereit ist. Bekommt der Prozess vom Scheduler einen Prozessor zugewiesen, so wechselt dieser in den Zustand “laufend“ und beginnt das Programm abzuarbeiten, bis der Scheduler dem Prozess die Prozessorzeit entzieht. Danach wechselt dieser wieder in den Zustand “bereit“. Es gibt jedoch auch die Möglichkeit, dass ein Prozess blockiert wird und demnach vom Zustand “laufend“ zum Zustand “blockiert“ wechselt. Dies ist mit den Synchronisationswerkzeugen möglich, wie zum Beispiel mit dem Mutex, der das gleichzeitige Lesen und Schreiben eines Speicherplatzes verhindert. Wird der Mutex aufgehoben, so nimmt der Prozess erneut den Zustand “bereit“ an. Des Weiteren kann das Blockieren durch das Warten auf Eingabe- / Ausgabe-Ereignisse hervorgerufen werden. In den Zustand “Zombie“ kann ein Kindprozess bei seiner Beendigung kommen. Solange der Elternprozess noch nicht beendet ist, werden die Ressourcen vom Kindprozess aufbewahrt.

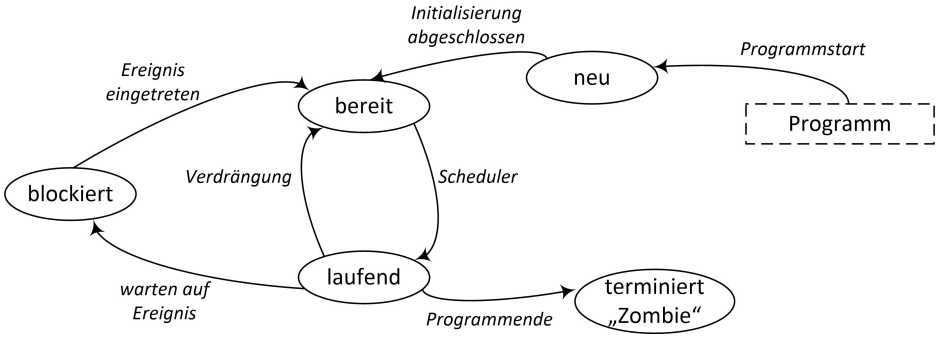

Abbildung 2.6: Zustände der Prozesse

Jedem Prozess wird vom Betriebssystem ein Prozess-Kontext zugeordnet, vgl. [5, Seite 79 ff], in dem viele Informationen stehen. Darin enthalten sind zum Beispiel die Daten, die vom Betriebssystem verwaltet werden und die Inhalte der Hardware-Register, wie der Befehlszähler. Zusätzlich ist der Stack darin enthalten, der als Inhalt die Variablen und die Rücksprungadressen hat. Muss ein aktiver Prozess die CPU abgeben, so wird der Prozess-Kontext vom Betriebssystem aufbewahrt. Dies ist für das Kontext-Switching notwendig. Neben dem Prozess-Kontext bekommt jeder Prozess eine eindeutige Identifikationsnummer (PID) vom Betriebssystem zugewiesen.

Threads sind nebenläufige Ausführungseinheiten innerhalb eines Prozesses. Gibt es nur einen Thread in einem Prozess, so ist der Thread und der Prozess gleichbedeutend. Ein Prozess kann mehrere Threads beinhalten, die den gleichen Speicherbereich zur Verfügung haben und die gleichen Ressourcen übergeben bekommen. Daher werden Threads auch als als leichtgewichtige Prozesse bezeichnet, vgl. [5, Seite 83].

2.4 Scheduling

Scheduling ist ein Verfahren der Taskverwaltung, um die Zuteilung des Prozessors für ablaufbereite Tasks, die sich im Zustand “bereit“ befinden, zu steuern, vgl. [1, Seite 356 ff]. QNX benutzt das Echtzeitscheduling, der die Task entsprechend einplant, so dass die Zeitbedingungen eingehalten werden können, sofern die benötigte Prozessorzeit nicht die verfügbare Prozessorzeit überschreitet. Die Berechnung der Prozessorauslastung H ist in der Gleichung 2.1 zu sehen, vgl. [1, Seite 257].

$$H = \frac{\textit{Benötigte Prozessorzeit}}{\textit{Verfügbare Prozessorzeit}} \tag{2.1}$$

Bei periodischen Prozessen ist die Analyse für das Scheduling zur Einhaltung der Zeitbedingungen recht einfach, da diese nicht unkontrolliert auftreten und somit vorhersagbar sind. Besteht ein System nur aus periodischen Prozessen, so kann die Analyse der Prozessorauslastung mit der Formel 2.2 durchgeführt werden, um das geeignete Scheduling-Verfahren auszuwählen, vgl. [1, Seite 257].

$$H = \sum_{i=1}^{n} \frac{e_i}{p_i} \quad \text{mit } e_i\text{: Ausführungszeit, } p_i\text{: Periodendauer von Task i} \tag{2.2}$$

Treten im System zusätzlich aperiodische Prozesse auf, so muss darauf geachtet werden, dass alle in einer Periode auftretenden Prozesse die Zeit dieser Periode nicht überschreiten. Das bedeutet, dass genügend zeitliche Kapazitäten in einer Periode für die aperiodischen Prozesse freigehalten werden, um die Echtzeitfähigkeit einhalten zu können.

Echtzeitscheduling-Verfahren können unterschiedlichen Klassen zugeordnet werden, wie die Abbildung 2.7 zeigt. Im statischen Scheduling-Verfahren wird das zeitliche Verhalten aller Prozesse vor der Ausführung geplant und in eine Zuordnungstabelle geschrieben.

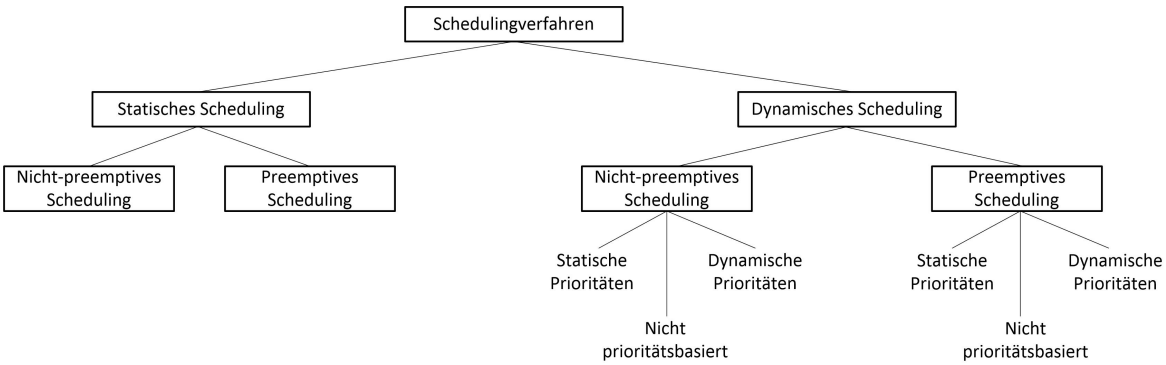

Abbildung 2.7: Unterschiedliche Klassen von Schedulingverfahren, vgl. [1, Seite 358]

2.4.1 FIFO

Prinzipiell funktioniert der FIFO-Scheduler wie eine Warteschlange. Die Bedeutung dieses Wortes ist “First In First Out“. Der Prozess, der sich zuerst meldet, bekommt als erster den Prozessor, vgl. [1, Seite 359 ff]. Beim Starten des Prozesses und nach der Initialisierung, wie Abbildung 2.6 zeigt, erreicht der Prozess den Zustand “bereit“ und wird im Speicher des Schedulers hinter dem Letzten abgelegt, so dass alle Prozesse in Reihenfolge abgearbeitet werden können. Dieser Scheduler-Speicher wird auch Queue genannt. Das FIFO-Verfahren ist ein dynamisches Scheduling und gehört im Allgemeinen zu der nicht-präemptiven Gruppe, so dass eine Unterbrechung der Prozesse durch das Betriebssystem nicht möglich ist. Die Prozesse können somit nicht vom Betriebssystem blockiert werden. In Echtzeitsystemen handelt der Scheduler präemptiv und basiert zusätzlich auf Prioritäten, vgl. [12, Seite 41]. In jeder Prioritätsgruppe gibt es eine eigene Warteschlange. Aufgrund des einfachen Algorithmus entsteht zur Laufzeitberechnung wenig Overhead, da weniger Kontextwechsel durch den Dispatcher auftreten, so dass wertvolle CPU-Zeit gespart werden kann. Ebenso ist die Umsetzung denkbar einfach. Der Nachteil dieses Schedulers ist die Nichtbeachtung der Zeitbedingungen. Somit ist dieser weniger geeignet für Echtzeitanwendungen. Allerdings ist dieser Scheduler in Echtzeitbetriebssystemen, wie zum Beispiel in QNX, vorhanden.

2.4.2 Round Robin

Das Round-Robin-Verfahren funktioniert im Grunde wie der FIFO-Scheduler. So ist dieser in normalen Betriebssystemen nicht-präemptiv und nicht prioritätsbasiert und in Echtzeitbetriebssystemen präemptiv und prioritätsbasiert, vgl. [5, Seite 119 ff]. Der Unterschied liegt darin, dass dieser mit einer Zeitscheibe versehen ist, so dass der laufende Prozess nach dem Ablauf der vorgegebenen Zeit den Zustand “blockiert“ annimmt und entsprechend den

Prozessor abgibt, wenn in der Warteschlange ein weiterer Prozess bereit ist. Ist dies nicht der Fall, so wechselt der blockierte Prozess wieder in den Zustand “laufend“ und beansprucht den Prozessor. Die Zeitscheibe wird auch Quantum genannt. Im Betriebssystem QNX ist das Quantum statisch mit dem Vierfachen der Taktperiode des Prozessors festgelegt und somit nicht änderbar, vgl. [12, Seite 42]. Sowohl das statische als auch das dynamische Quantum ist in den heutigen Betriebssystemen gleichverteilt vorhanden. Der Vorteil bei dem Round-Robin-Verfahren ist, dass die Zeitscheibe eine feste Zeit vorgibt, die ein Prozess nutzen darf, bevor dieser blockiert wird, so dass der nächste Prozess den Prozessor nutzt. Der blockierte Prozess reiht sich im Queue hinten ein und wartet, bis er wieder Laufzeit vom Scheduler erhält.

2.4.3 Sporadic

Aus periodischen Ereignissen kann ein Echtzeitnachweis erfolgen, da die Ereignisse zu jeder Periode den gleichen Rechenaufwand haben. Im Echtzeitbetrieb kommt es jedoch sehr selten vor, dass ausschließlich periodische Ereignisse auftreten. Üblicherweise treten zusätzlich aperiodische Ereignisse auf, die jeweils unterschiedliche Abarbeitungszeiten haben, so dass ein Echtzeitnachweis nicht erfolgen kann, vgl. [13]. Das Sporadic-Scheduling ist ein Verfahren, um aperiodische Ereignisse in periodische Ereignisse zu überführen. Folgende Parameter sind dem Scheduler zu übergeben, vgl. [12, Seite 42 ff]:

- **Initial budget**
  Damit ist die Bearbeitungszeit des Prozesses gemeint. Nach dem Verbrauch dieser Zeit, wird die Priorität herabgesetzt.
- **Replenishment period**
  Dies ist die maximale Prozessorzeit, die ein Prozess verbrauchen darf.
- **Normal priority**
  Mit dieser Priorität starten alle Prozesse. Dies ist die Priorität, wenn die Bearbeitungszeit noch nicht verbraucht ist.
- **Low priority**
  Die geringere Priorität wird dem Prozess zugeteilt, wenn dieser seine Abarbeitungszeit verbraucht hat und sich noch innerhalb der Prozessorzeit befindet.
- **Max number of pending replenishments**
  Dies beschreibt die maximale Wiederholung der Wiedervergabe der normalen Priorität nach der Prozessorzeit, um einen zu großen Overhead zu vermeiden.

Die beiden Zeitfenster sind die Parameter für die Prozessorzeit (*Replenishment period*) und für die Abarbeitungszeit (*Initial budget*), in der der Prozess mit seiner normalen Priorität läuft. Somit bekommt jedes Ereignis das gleiche Zeitfenster und die aperiodischen Ereignisse werden vom System wie periodische Ereignisse behandelt. Des Weiteren nutzt dieser Scheduler das Rate-Monotonic-Verfahren zur dynamischen Vergabe der Priorität, wobei die Prozesse in ihrer Abarbeitungszeit in die normale Priorität und nach Ablauf der Bearbeitungszeit in die geringere Priorität versetzt werden. Deshalb wird als weiterer Parameter dem Scheduler die normale (Normal priority) und die geringere Priorität (Low priority) mitgegeben. Der letzte Parameter gibt die Anzahl der Auffrischungen an, in der er nach dem Ablauf der Prozesszeit wieder die normale Priorität bekommt.

2.5 Analog-Digital-Wandler / Digital-Analog-Wandler

Die Analog/Digital-Wandlung (ADC) und die Digital/Analog-Wandlung (DAC) gehören zum Themenbereich der digitalen Signalverarbeitung, vgl. [3, Seite 354 ff]. In der heutigen Technik werden überwiegend digitale Signale verarbeitet, da diese Art der Verarbeitung im Vergleich zur Analogtechnik Vorteile bietet, vgl. [4]:

- keine Bauteiletoleranzen
- keine äußeren Einflüsse auf Bauteile, die Signale verfälschen
- Reproduzierbarkeit
- Flexibilität
- geringere Kosten
- Algorithmen implementierbar, die in der analogen Technik nur schwer realisierbar sind

Durch eine endliche Anzahl an Werten des digitalen Signals und durch einen endlichen Wertebereich der Werte selbst ist die numerische Verarbeitung der digitalen Signale möglich. In den meisten Fällen wird zum Schluss wieder ein analoges Signal benötigt. Die Verarbeitungskette zeigt Abbildung 2.9. Zu Beginn der Umwandlung liegt das analoge Signal als zeitkontinuierliches und wertkontinuierliches Signal ($s(t)$) vor, welches in Abbildung 2.8 gezeigt ist. Der erste Schritt ist die Zeitquantisierung. Dabei wird erst das Signal abgetastet. Daraus entsteht ein zeitdiskretes und wertkontunierliches Signal ($s[k]$). Daraufhin wird das abgetastete Signal bis zur nächsten Abtastung gehalten. Dies wird auch "Sample & Hold" genannt und es entsteht das Signal $s_q(t)$. Der zweite Schritt ist die A/D-Wandlung, in der die Werte mit einer vorgegebenen Auflösung quantisiert und in Binärformat umgewandelt werden ($s_q[k]$) und dem Rechner seriell oder parallel übergeben werden. $s_q[k]$ ist das digitalisierte Signal. Wird beispielsweise eine 4-Bit ($n = 4$) Auflösung verwendet, so wird die

maximale Eingangspannung U_{emax} in $2^n = 2^4 = 16$ Stufen zerlegt. U_{emax} ist die so genannte Full Scale Range (FSR) und beschreibt den gesamten Spannungsbereich, der entsprechend der Auflösung zerlegt wird. Die Spannungswerte müssen nicht nur positiv sein. Diese können sich auch vom negativen bis zum positiven Bereich erstrecken, wodurch der gesamte Spannungsbereich der Peak-to-Peak-Wert ist. In codierter Darstellung ist der maximale Wert nicht darstellbar, da die größte binäre Zahl $2^n - 1$ ist. Demnach ist der maximal darstellbare Spannungswert (U_{dmax}) die maximale Eingangsspannung, reduziert durch das Quantisierungsintervall (ΔU_q), das durch die Gleichung 2.3 beschrieben wird.

$$U_{dmax} = U_{emax} - \Delta U_q \tag{2.3}$$

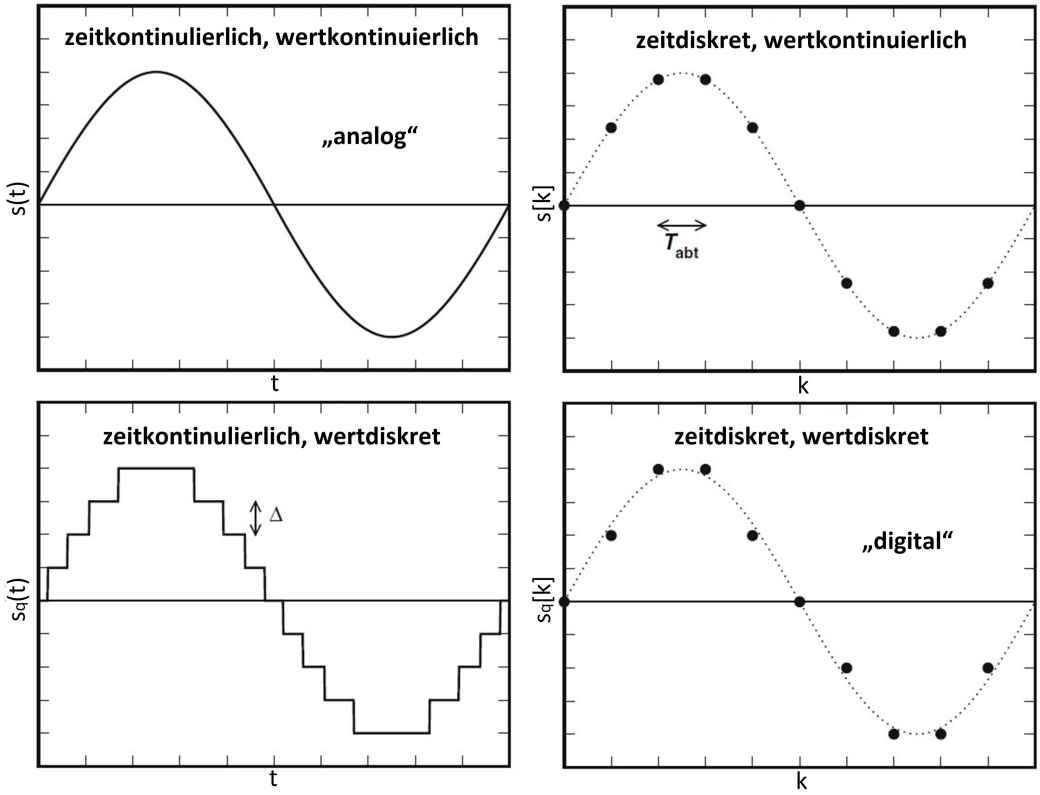

Abbildung 2.8: Darstellung der Signalarten in der Signalverarbeitungskette, vgl. [3, Seite 355]

Somit hat der Rechner die Möglichkeit, die Daten zu bearbeiten. Um das Signal im Anschluss zu analogisieren, kommt der D/A-Wandler zum Einsatz, der die Werte decodiert. Der Wert und die Zeit bleiben diskret. Erst nach einem Tiefpassfilter, der das Signal interpoliert, wird daraus wieder ein analoges Signal.

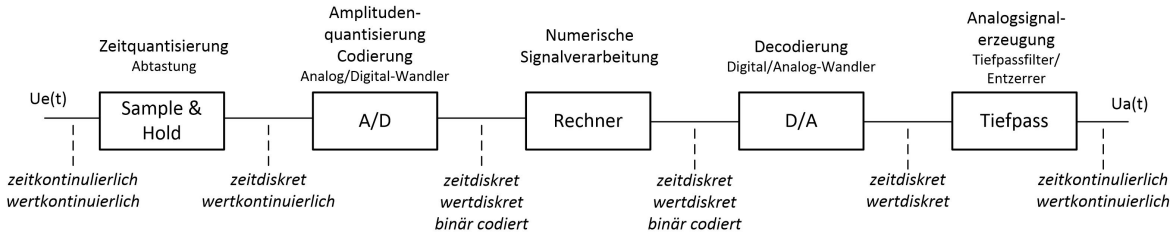

Abbildung 2.9: Blockschaltbild einer Signalverarbeitungskette mit den dazugehörigen Signalarten, vgl. [4, Seite 4]

Bei der Digitalisierung eines Signals ist das Einhalten des Abtasttheorems wichtig, da das Signal sonst am Ende der Verarbeitungskette nicht wieder in ein analoges Signal umgewandelt werden kann. Bei der Abtastung wird das Signal im Spektrum periodisch fortgesetzt. Wird ein Signal mit zu großem Abstand abgetastet, so entstehen bei diesen Fortsetzungen Überlappungen, die mit dem Tiefpass nicht mehr herausgefiltert werden können. Das Abtasttheorem ist wie folgt definiert:

$$f_A > 2 \cdot f_{Smax} \tag{2.4}$$

Die Abtastfrequenz (f_A) muss größer als das doppelte der maximal vorkommenden Frequenz des Signals (f_{Smax}) sein. Wird das Abtasttheorem verletzt, treten nichtlineare Verzerrungen durch die Überlappung auf. Das Resultat dieser Verletzung ist auch als Alias-Effekt bekannt.

Die bislang beschriebene Umwandlung bezieht sich allerdings nur auf eine ideale Abtastung. In realen Systemen lässt sich der Alias-Effekt nicht vermeiden, weil zum einen bei der Interpolation zur Rückgewinnung kein idealer Tiefpass eingesetzt werden kann, da dieser keine senkrecht ansteigende Flanke hat. Dennoch kann der Effekt durch geeignete Maßnahmen, wie zum Beispiel der Antialias-Filter, bis zu einem annehmbaren Maß reduziert werden, so dass die Rückgewinnung mit hinnehmbaren Verlusten durchgeführt werden kann. Zum anderen wird die Abtastung im idealen System durch Dirac-Impulse realisiert. Dies ist in realen Systemen ebenfalls nicht möglich, da die Impulse aus physikalischen Gegebenheiten eine gewisse Höhe und Breite haben. Weitere Probleme werden durch Quantisierungsrauschen verursacht, da die Bit-Codierung nur eine endliche Auflösung hat. Je nach Qualität der Rückgewinnung, die für ein System erforderlich ist, ist die Wahl einer geeigneten Auflösung notwendig, denn generell gilt auch aus wirtschaftlicher Sicht: “So groß wie nötig und so gering wie möglich“.

2.6 Interrupt / Polling

In den heutigen Betriebssystemen werden sowohl Software- als auch Hardwareinterrupts benutzt. Das Betriebssystem muss während des normales Betriebs auf plötzliche Anfragen möglichst schnell reagieren. Diese Unterbrechungsanfragen können dabei von externer Hardware, wie zum Beispiel der Netzwerkkarte, dem CD-Laufwerk oder der Festplatte, aufgerufen werden, die als asynchrone Interrupts definiert sind. Das bedeutet, dass die Unterbrechungsanfragen weder vorhersehbar noch reproduzierbar sind. Diese Anfragen nennt man Hardwareinterrupt. Daneben gibt es die Möglichkeit, dass diese Anfragen von einem Programm oder dem System selbst aufgerufen werden, welche Systemcalls genannt werden. Eine weitere Möglichkeit der Unterbrechung sind die Exceptions (*Ausnahmen*). Diese werden vom System aufgerufen, wenn schwerwiegende Fehler auftreten, um einen Systemausfall zu vermeiden und um die Möglichkeit zu haben, entsprechend darauf zu reagieren. Sowohl die Softwareinterrupts als auch die Exceptions sind synchrone Interrupts, die vorhersehbar und reproduzierbar sind, vgl. [5, Seite 52 ff]. Die Interruptklassifizierung zeigt die Abbildung 2.10.

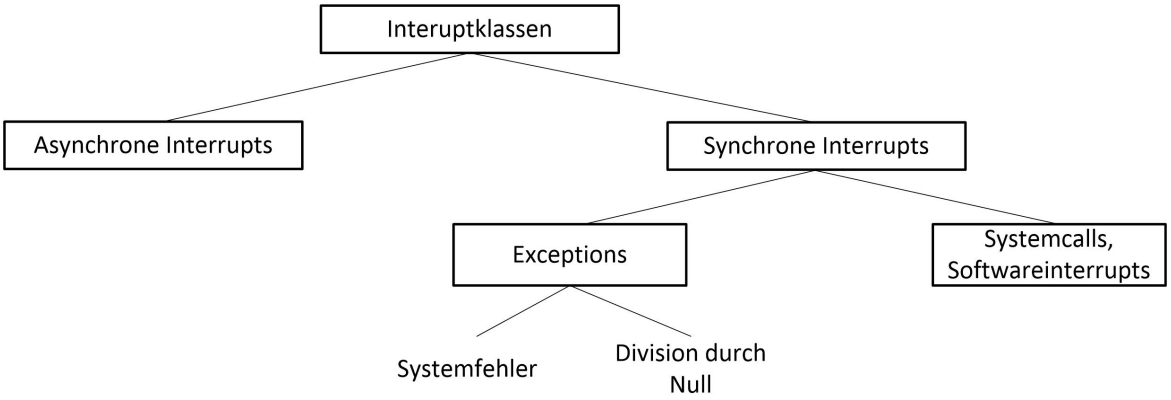

Abbildung 2.10: Interruptklassen, vgl. [5, Seite 52]

Bei allen Interruptklassen wird nach einer Unterbrechungsanfrage ein extra Programm-Code abgearbeitet, um entsprechend auf dieses Ereignis zu reagieren. Dieser Programm-Code wird Interrupt-Service-Routine genannt. Dabei ist es unter Umständen wichtig, dass dieser Ablauf nicht durch eine weitere Anfrage unterbrochen wird. Es ist möglich, durch die so genannte Maskierung bestimmte Hardwareinterrupts zu deaktivieren, um eine Unterbrechung zu vermeiden. Diese wird üblicherweise nach Bearbeitung dieses kritischen Bereichs wieder aktiviert. Des Weiteren werden beim Systemstart Interrupt-Level definiert, so dass bei gleichzeitigem Auftreten der höher priorisierte Interrupt erst behandelt wird und bereits laufende ISRs nicht unterbrochen werden.

Da die Hardware überwiegend keine Interrupts direkt an die CPU senden kann, gibt es die Interrupt-Controller als Zwischeneinheit. Dieser empfängt das Signal von der Hardware, welcher als Interrupt-Request (IRQ) bekannt ist und leitet den IRQ als Unterbrechung zur CPU weiter. Die Kommunikation wird durch eine Interrupt-Vektor-Tabelle durchgeführt. Diese dient der zentralen Aufnahme von ISRs und ist an einer vordefinierten Stelle des Kernels gespeichert. Alle möglichen Interruptarten bekommen einen Index zugeteilt.

Polling ist ebenfalls eine Variante, um auf Signale zu reagieren. Allerdings fragt diese regelmäßig den Status ab, um Unterbrechungsanfragen zu registrieren, auch “Busy-Waiting“ genannt. Dieses Verfahren ist einfach zu implementieren. Dennoch überwiegt der Nachteil, dass das Verfahren ständig Prozessorzeit verbraucht. Üblicherweise wird Polling in der Softwareentwicklung vermieden.

3 Systembeschreibung

Zur Entwicklung und Durchführung der Arbeit stehen die Geräte des Automatisierungslabors zur Verfügung. Darin enthalten ist das Target-System als 19“-Baugruppenträgerrechner mit dem Board “Hercules II EBX“ des Unternehmens Diamond Systems Corporation. Auf einem Flashspeicher ist QNX Neutrino vorinstalliert und somit betriebsbereit. Für die AD/DA-Umsetzung wird ein Funktionsgenerator und zur Überprüfung ein Oszilloskop, welches direkt auf dem Host-Rechner betrieben wird, verwendet. Die Entwicklungsumgebung QNX Momentics IDE basiert auf Eclipse und wird über Ethernet ebenfalls auf einem Host-Rechner betrieben. Der gesamte Aufbau des Systems ist in Abbildung 3.1 zu sehen.

Abbildung 3.1: Aufbau des Echtzeitsystems

3.1 Hardware

Die Hardware ist durch das Labor der Automatisierungstechnik vorgegeben. Dabei handelt es sich um ein eingebettetes System des Unternehmens “Diamond Systems Corporation“, welches in einem 19“-Baugruppenträger verbaut ist. Dabei sind die Ein- und Ausgänge, die für das Labor benötigt werden, nach außen geführt, wie zum Beispiel die digitalen und analogen Ein- und Ausgänge und die Ethernet-Schnittstelle. Des Weiteren wird ein kleines Oszilloskop für die Messung genutzt, welches die Signale über die Software “Multi Channel Software“ auf dem Host-Rechner anzeigt. Um das analoge Eingangssignal zu erzeugen, wird ein Funktionsgenerator der Firma Hewlett Packard verwendet.

3.1.1 Hercules II EBX

Das Board “Hercules II EBX“ aus Abbildung 3.2 stammt von dem Unternehmen “Diamond Systems Corporation“ und ist für den Zweck der eingebetteten Systeme entwickelt worden, vgl. [6]. Der Zusatz EBX steht für “Embedded Board eXpandable“.

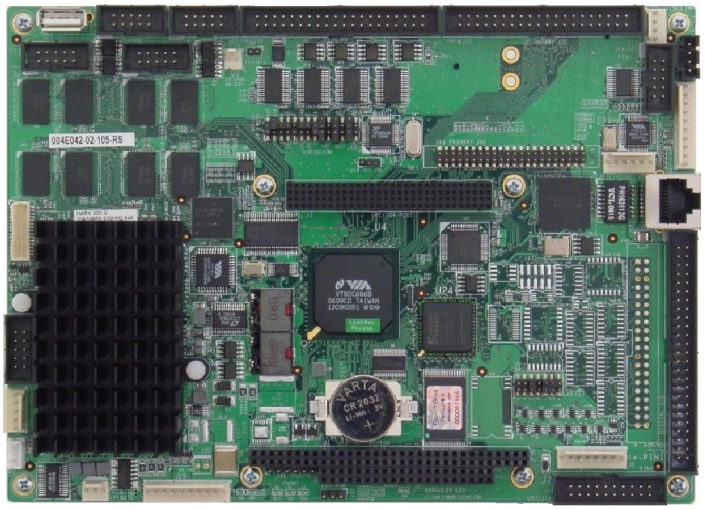

Abbildung 3.2: Darstellung des Hercules II EBX-Board

Auf dem Board sind Komponenten enthalten, die im Folgenden aufgelistet werden:

- CPU
  - Pentium-3 Via Mark Integrated Processor
  - 512 MByte SDRAM
  - 100 MHz Memory Bus

  - 2 MByte und 16-Bit breiter, integrierter Flash-Speicher für das BIOS und weitere Programme
  - 2D & 3D 128-bit Architecture VGA Video Graphics Engine
  - 33 MHz PCI Bus

- Core PC Chipset (mit Memory Controller, PCI-Schnittstelle und ISA-Schnittstelle)
- I/O-Anschlüsse
  - 4 serielle Ports mit maximal 115,2 kbaud
  - 4 USB 1.1 Ports
  - 4 USB 2.0 Ports
  - 2 IDE Schnittstellen (Standard 40-Pin IDE und 44-Pin Version für Notebook-Laufwerke)
  - SSD direkt auf dem Board anschließbar
  - 10/100 BaseT full-duplex PCI Bus Mastering Ethernet
  - IrDA Port (dafür wird ein kombiniertes Sende- und Empfangsgerät benötigt)
  - PS/2 Tastatur- und Maus-Ports
  - LEDs
  - Lautsprecher-Schnittstelle
- Analog Input/Output
  - 16-Bit Auflösung für Eingangssignale, 12-Bit Auflösung für Ausgangssignale
  - 250 kHz maximale A/D Abtastrate
  - einstellbarer Eingangsspannungsbereich mit maximalem Bereich von $\pm$10 V für Ein- und Ausgang
  - bipolarer und unipolarer Eingangsspannungsbereich
  - interner und externer A/D Trigger
  - 2048-Abtastwerte-Kapazität FIFO
- Digital Input/Output
  - 40 programmierbare digitale Ein- und Ausgänge mit 3.3 V und 5 V
  - - 8 / + 12 mA maximaler Ausgangsstrom
  - Pull-up / Pull-down Widerstand wählbar
- Counter / Timer
  - 24-Bit Counter/Timer für die A/D Abtastrate
  - 16-bit Counter/Timer für die Benutzerfunktionen des Counters und des Timers
  - sowohl interner als auch externer Takt möglich
  - 4 programmierbare Pulsweitenmodulation-Signale

- System-Eigenschaften
  - Plug and Play BIOS mit automatischer IDE-Erkennung
  - 1 Compact-Flash-Schnittstelle
  - System-Wiederherstellung-ROM im Falle eines BIOS-Fehlers
  - On-Board Lithium Backup Batterie für den Echtzeittakt und dem CMOS RAM
  - Board in ATX-Format
  - programmierbarer Watchdog-Timer
- Audio
  - Realtek AC97 Audio Codecs
  - Stereo Line In
  - Stereo Line Out
  - Mono Mic In
  - Stereo Internal Line In

3.1.2 Oszilloskop

Das Oszilloskop Handyscope 3 (HS3) der Firma TiePie Engineering, in Abbildung 3.3 zu sehen, ist sehr gut für diese Laboraufgabe der A/D- und D/A-Echtzeitmessung geeignet. Es ist kostengünstig und bietet alle benötigten Funktionen zur Aufnahme dieser Messung. Durch die Größe des Geräts ist es portabel und besitzt zwei Eingänge und einen Ausgang, mit dem verschiedene Signalformen generiert werden können, vgl. [14].

Abbildung 3.3:
Oszilloskop Handyscope 3 der Firma TiePie Engineering

CH1 und CH2 sind BNC-Buchsen und bieten die Möglichkeit, analoge Eingangssignale zu messen. Dabei sind die Außenseiten der Eingangsbuchsen, sowie die Ausgangsbuchse, mit dem HS3-Gerät geerdet. Die Stromversorgung und der Datentransport sind über ein USB-Kabel realisiert, das an dem Host-Rechner angeschlossen werden kann.

3.1.3 Funktionsgenerator

Der Funktionsgenerator der Firma Hewlett Packard wird zur Erzeugung eines analogen Eingangssignals für das System verwendet. Der Generator bietet die Möglichkeit verschiedene Signalformen auszugeben. Darunter kann das Gerät Sinus-Signale, Rechteck-Signale, Impulse, Dreieck-Signale und Sägezahn-Signale erzeugen, die im Bedienfeld anhand eines Drehknopfes eingestellt werden können. Des Weiteren kann der Generator Frequenzen im Bereich 0.0005 Hz bis 5 MHz für alle Signale erzeugen, vgl. [15, Seite 2]

Abbildung 3.4:
Funktionsgenerator 3310A von Hewlett Packard

Die Erzeugung der Frequenz wird in Dekaden mit einem Drehknopf eingestellt. Ein großer Drehknopf, der sich ganz links am Bedienfeld befindet, dient zur genauen Einstellung der Frequenz. Die so eingestellte Zahl entspricht dem Faktor, mit dem die Dekade multipliziert wird. Daraus ergibt sich die Frequenz, mit der das Signal ausgegeben wird. Eine weitere Einstellungsmöglichkeit ist der Offset und die Amplitude, die anhand weiterer Drehknöpfe eingestellt werden können.

3.1.4 Hostsystem

Das Host-System ist ein gewöhnlicher PC mit Windows 8 als Betriebssystem. Zur Kommunikation mit dem Target-Rechner besteht eine Ethernet-Verbindung. Auf diesem Rechner ist die Entwicklungsumgebung QNX Momentics installiert.

3.1.5 Digitale I/O-Panel

Der Digital-Input-Output-Panel (DIO-Panel) ist eine Schnittstelle, die von der Werkstatt der Hochschule für Angewandte Wissenschaften Hamburg (HAW) für das Hercules II EBX-Board entwickelt und gebaut wurde. Dabei wird die Schnittstelle mit einem 40-poligen IDE-Flachbandkabel am Board verbunden. Das DIO-Panel bietet die Möglichkeit, Port C0 bis C7 als Eingangssignal mit Hilfe von Schaltern zu nutzen. Die Schalter sind über eine entsprechende elektronische Verschaltung entprellt, so dass bei Betätigung nur ein High- oder Low-Signal gesendet wird.

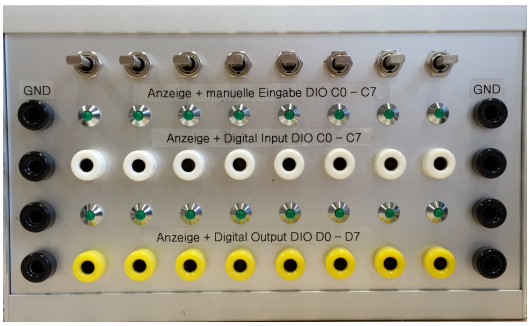

Abbildung 3.5:
digitale Steuereinheit mit integrierten Schaltern

Die Schalter am Panel dienen nur zum Aufschalten eines High- oder Low-Signals an den C-Ports. Die D-Ports können damit nicht bedient werden. Des Weiteren besteht die Möglichkeit, D0 bis D7 als digitalen Ausgang oder Eingang zu nutzen. Wird der D-Port als Eingang verwendet, so muss ein externes Signal für das High- und Low-Signal genutzt werden. Sowohl die Eingangspins als auch die Ausgangspins haben jeweils eine LED, die bei einem High-Signal leuchten.

3.2 Software

Die benötigte Software ist durch das Automatisierungslabor vorgegeben, indem bereits Laboraufgaben im Bereich Betriebssysteme durchgeführt werden. Für die A/D- und D/A-Messung wird ein Oszilloskop mit Softwarelösung direkt auf dem Host-Rechner betrieben. Für die Entwicklungsumgebung wird das QNX Momentics IDE verwendet, welches auf dem Host-Rechner installiert wird. Als Echtzeitbetriebssystem wird QNX Neutrino auf dem Zielrechner verwendet. Um spezielle Funktionen des Boards zu nutzen, wird zur Entwicklung der Treiber “Universal Driver“ von Diamond Systems genutzt.

3.2.1 QNX Neutrino

QNX Neutrino ist ein Echtzeitbetriebssystem, welches von Diamond System auf einem Flashspeicher vorinstalliert ist. Das Betriebssystem läuft auf dem Hercules II EBX-Board. Eine genaue Beschreibung von QNX ist im Grundlagenkapitel zu finden.

3.2.2 QNX Momentics IDE

QNX Momentics IDE (Integrated Development Environment) ist die Entwicklungsumgebung von QNX. Dabei wurde die Entwicklungssoftware Eclipse als Grundlage genutzt und an die Bedürfnisse von QNX angepasst. Momentics wurde entwickelt, um Projekte für Zielrechner zu programmieren, auf denen QNX Neutrino RTOS (Realtime-Operation-System) installiert ist. Diese Software wird üblicherweise auf einem Host-Rechner betrieben, um Projekte von der Ferne auf das QNX-System zu entwickeln. Dabei ist die Kompatibilität mit vielen Betriebssystemen gegeben. Darunter ist Windows XP, 7, 8, Linux-Systeme und QNX Neutrino. Dennoch ist es möglich, dass die Entwicklungsumgebung direkt auf dem Zielrechner betrieben wird, wenn ausreichend Ressourcen vorhanden sind. Die Entwicklungsumgebung hat folgende Eigenschaften und Funktionen:

- Organisation der Strukturen, Projekte und Dateien
- Bearbeitung des Programmcodes
- Projekte mit einem Team bearbeiten
- Kompilierung, Ausführung und Debugging eines Projekts auf einem eingebetteten Systems
- Erstellen von OS und Flash-Images
- Analyse und Optimierung des Systems

3.2.3 Multi Channel Software

Die Mess-Software der Firma TiePie Engineering dient der Visualisierung und Verarbeitung der Signale, die von dem HS3 dem Host-Rechner geliefert werden. Multi Channel Software hat viele Möglichkeiten, das Signal zu messen. Zum einen kann es normal in Abhängigkeit der Zeit (Yt) angezeigt werden, zum anderen kann es in Abhängigkeit eines anderen Signals (XY) angezeigt werden. Eine weitere Möglichkeit der Multi Channel Software ist das Anzeigen des Spektrums eines Signals. Um Zeitdifferenzen oder Spannungswertdifferenzen berechnen zu lassen, kann die Software anhand von horizontalen und vertikalen Cursor die Differenzen berechnen, vgl. [16].

3.2.4 Universal Driver von Diamond Systems

Das Unternehmen Diamond Systems bietet zur Entwicklung einen Treiber an, der als Schnittstelle zwischen dem Hercules II EBX-Board und der Entwicklungsumgebung dient, vgl. [17]. Darin enthalten sind viele Funktionen, die zur Bearbeitung von den Ein- und Ausgängen, sowie für die Timer und den Watchdog-Timer zur Verfügung stehen. Dieser Treiber “UD-6.02“ muss als Bibliothek eingebunden werden. Dazugehörig gibt es eine Header-Datei “dscud.h“, die ebenfalls in das Projekt kopiert werden muss. Um die Funktionen des Treibers besser zu verstehen, gibt es einige Beispiel-Codes, wie zum Beispiel zur A/D-Wandlung, D/A-Wandlung, User-Interrupts und Watchdog. Für die Nutzung des Treibers sind für jedes Programm 4 Funktionen wichtig. Zum einen müssen der Treiber und das Board initialisiert werden, da Diamond Systems verschiedene Boards anbietet. Das geschieht mit den Funktionen “dscinit“ und “dscinitBoard“. Zum anderen müssen die Ressourcen und das Board zum Schluss des Programm wieder freigegeben werden. Dies übernehmen die Funktionen “dscFreeBoard“ und “dscFree“, vgl. [17, Seite 16].

4 Konzeption / Design

Das Ziel ist, eine Laboraufgabe für den Masterstudiengang der Automatisierung zum besseren Verständnis von Echtzeitsystemen zu entwickeln. Zur Umsetzung eines Echtzeitsystems werden ein Echtzeitbetriebssystem und ein Aufbau, der in Echtzeit abgearbeitet werden muss, benötigt. Da die weiteren Laboraufgaben auf QNX basieren und bereits alle Laborplätze mit einem eingebetteten System, worauf QNX installiert ist, ausgestattet sind, kann dieses ebenso für die Echtzeitaufgabe genutzt werden. Zudem bietet QNX die geforderten Bedingungen, wie unter anderem eine Entwicklungsumgebung, mit der das Projekt von einem Host-Rechner direkt auf das Target-System implementiert werden kann. Ebenso wird das eingebettete System mit dem Board von Diamond System “Hercules II EBX“ verwendet, da es die Möglichkeit bietet, mit den analogen Ein- und Ausgängen A/D-D/A-Umwandlungen zu generieren, um so ein Echtzeitsystem näher zu beleuchten und zu untersuchen. Zur Steuerung des Programms können digitale I/Os verwendet werden. Zur Erzeugung einer Eingangsspannung wird ein Funktionsgenerator der Firma Hewlett Packard verwendet, da dieses Gerät in einer ausreichenden Stückzahl im Labor vorhanden ist. Da das eingebettete System zur Entwicklung der Software zu wenig Ressourcen zur Verfügung hat, wird ein Host-Rechner mit der Entwicklungsumgebung QNX Momentics IDE benötigt. Mit dem Generator kann die Frequenz und die Form des Signals eingestellt werden. Zusammengefasst werden folgende Bedingungen an das Projekt gestellt:

- Echtzeitbetriebssystem
- Echtzeit-Scheduler
- analoge Ein- und Ausgänge
- A/D-D/A-Controller
- digitale I/Os
- Spannungserzeugung und Frequenzeinstellung

Die Idee ist es, zwei Prozesse stabil auf dem System auszuführen. Dabei soll der eine Prozess die A/D-Umwandlungen vornehmen und in einem Puffer speichern. Der A/D-Wandler hat eine 16-Bit Auflösung. Der zweite Prozess übernimmt die Werte des A/D-Wandlers und speichert diese in einem eigenen Puffer. Dabei muss der D/A-Wandler diese Werte entsprechend seiner 12-Bit Auflösung umrechnen, bevor diese gespeichert und danach ausgegeben werden können. Diese beiden Prozesse werden in dem Programm als Threads implementiert. Läuft die A/D-D/A-Umwandlung stabil, so kann davon ausgegangen werden, dass jeder

Prozess ausreichend Prozessorzeit zur Verfügung hat. Um die Echtzeitfähigkeit zu prüfen, soll das Programm und dessen Threads mit den drei unterschiedlichen Scheduler, die QNX anbietet, ausgeführt werden. Die drei Scheduling-Verfahren sind FIFO, Round-Robin und Sporadic, die bereits im Grundlagenkapitel beschrieben sind. Bei der Betätigung des CO-Schalters (C null) soll alle 100 ms ein Thread erstellt werden, der in einer Dauerschleife eine Variable hochzählt, um fortwährend mehr Prozessorzeit in Anspruch zu nehmen. Sind genügend Counter-Threads aktiv, so wird ausreichend Prozessorzeit beansprucht. Dies hat zur Folge, dass der A/D-Thread bei dem Entleeren seines FIFO's nicht mehr genug Zeit hat und dieser überläuft. Beim Überlauf sollen die Threads abbrechen. Zum Schluss soll die Anzahl der Counter-Threads auf die Konsole ausgegeben werden, um somit die Effektivität und die Qualität der unterschiedlichen Scheduling-Verfahren zu prüfen. Die Vorgehensweise ist als Aktivitätsdiagramm in Abbildung 4.1 und 4.2 zu sehen.

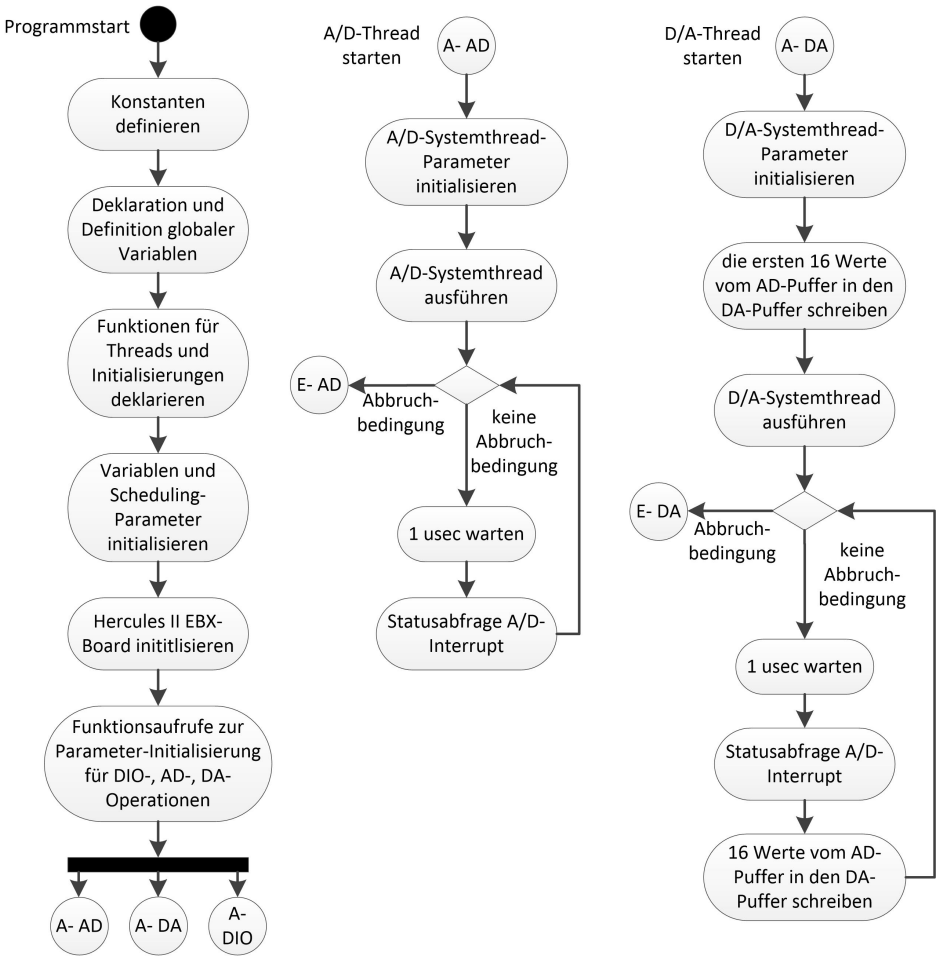

Abbildung 4.1: Aktivitätsdiagramm des Main-, A/D-, D/A-Threads

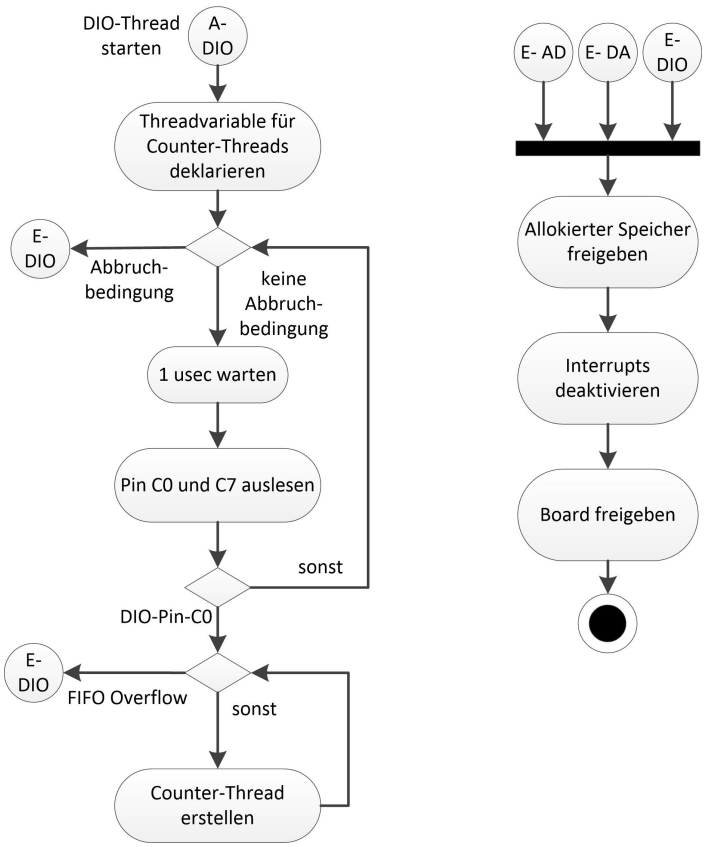

Abbildung 4.2: Aktivitätsdiagramm des Main-, DIO-Threads

5 Realisierung

Das Kapitel der Realisierung beinhaltet alle Schritte, die notwendig sind, um das Projekt umzusetzen. Dabei muss die Entwicklungsumgebung QNX Momentics IDE auf dem Host-System installiert werden. Zur Kommunikation zwischen dem Target- und dem Host-System muss die Netzwerkschnittstelle konfiguriert werden, damit vom Netzwerk aus auf das eingebettete System das Projekt entwickelt werden kann. Nach dem Erstellen des QNX-Projektes unter Momentics und dem Einbinden des Target-Systems und des Treibers “Universal Driver“ von Diamond Systems wird das Projekt in der Programmiersprache C entwickelt. Dies erfordert, dass alle wichtigen Header-Dateien eingebunden sind. Im Anschluss werden die Defines gesetzt, damit der Code lesbar bleibt, und alle benötigten globalen Variablen und Funktionen werden definiert. Das Main-Programm ist verantwortlich für das Erstellen der A/D-, D/A- und DIO-Threads, die ebenfalls entwickelt werden. Zum Ende des Programms werden die Ressourcen und das Board wieder freigegeben.

5.1 Konfiguration eines neuen Projektes unter QNX Momentics IDE

Da das Echtzeitbetriebssystem QNX Neutrino bereits auf dem eingebetteten System installiert ist und die Netzwerkeinstellung mit der IP-Adresse 192.168.57.205 eingerichtet ist, wird lediglich die QNX Software Development Platform (SDP) auf dem Host-System installiert. Dieses Software-Paket beinhaltet QNX Momentics IDE und alle wichtigen Komponenten, wie zum Beispiel den Compiler, den Linker und die Bibliotheken, die zur Entwicklung eines Programms notwendig sind.

Zu Beginn des Projektes muss ein QNX-Projekt unter Momentics erstellt werden. In dem Assistenten, der sich daraufhin öffnet und in Abbildung 5.1 zu sehen ist, wird der Name des Projektes angegeben. Alle weiteren Einstellungen können beibehalten werden. Entsprechend wird der Pfad zur Entwicklung dieses Projekts nicht verändert und befindet sich in C:\ide-4.5-workspace\AD_AD_CONV_QNX. Darin enthalten sind die Quellcodes, die Bibliotheken und die Header-Dateien, die ebenfalls in der Dateistruktur-Ansicht in der Entwicklungsumgebung angezeigt werden.

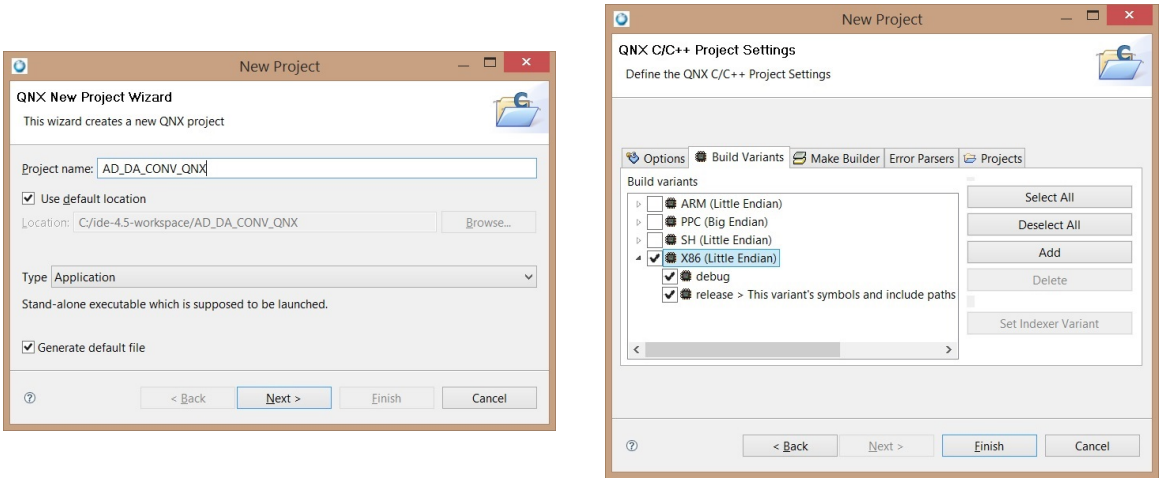

Abbildung 5.1: Assistent zum Erstellen des Projektes Seite 1, 2

Mit dem Feld “Next“ werden die Einstellungen der ersten Seite des Assistenten bestätigt. Daraufhin erscheint eine zweite Seite in der im Reiter “Build Variants“ nur die CPU-Architektur “X86“ des Ziel-Rechners eingestellt werden muss. Dabei ist darauf zu achten, dass beide Unterpunkte aktiviert sind. Des Weiteren können alle voreingestellten Parameter belassen werden. Mit “Finish“ werden die Einstellungen übernommen und das Projekt mit einem automatisch generierten Code von QNX erstellt.

Damit die Kommunikation über Ethernet mit dem Ziel-Rechner funktioniert, muss der Daemon “qconn“ auf dem eingebetteten System gestartet werden. Dieses Programm bietet die benötigten Dienste an, damit die Entwicklungsumgebung von dem Host-Rechner auf dem QNX-Rechner zugreifen kann. Dieser ermöglicht auch die Anzeige der Systeminformationen des eingebetteten Rechner auf Momentics. Der nächste Schritt ist die Einbindung des QNX-Rechners in die Entwicklungsumgebung. Dafür benötigt Momentics die IP-Adresse des Ziel-Rechners. Um diese Einstellungen vorzunehmen, wird in das Fenster “QNX System Information perspective“ gewechselt. Mit einem Rechtsklick im Feld “Target Navigator“ kann im Dropdown-Menü ein neues Ziel angelegt werden. Im Assistenten wird ein Name und die IP-Adresse des Ziels angegeben, damit der Entwicklungsumgebung das Ziel bekannt ist. Dies ist in Abbildung 5.2 zu sehen. Die Meldung im Assistenten “Target exists“ wird angezeigt, da das Ziel bereits angelegt wurde. Diese Informationen können der PDF-Datei [18] von QNX entnommen werden.

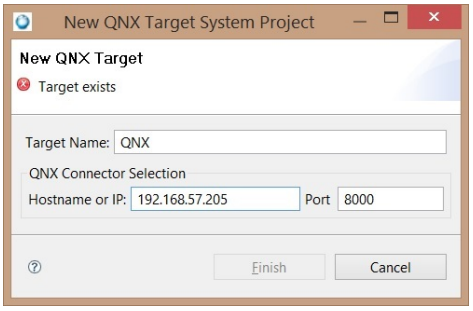

Abbildung 5.2: Assistent zur Erstellung eines neuen Ziels

Um die Universal-Driver-Bibliothek und die Mathe-Bibliothek einzubinden, werden mit Rechts-Klick auf das erstellte Projekt die Eigenschaften ausgewählt. Im Unterpunkt “QNX C/C++ Project“ unter dem Reiter “Linker“ können diese Bibliotheken in der Kategorie “Extra libraries“ hinzugefügt werden. Dieses Fenster ist in Abbildung 5.3 zu sehen. Der Linker dient zur Zusammenführung aller Objekt-Dateien eines Programms, die vom Compiler bereitgestellt werden. Zudem ordnet der Linker die relativen Adressen, die vom Compiler vorliegen, in absolute Adressen um.

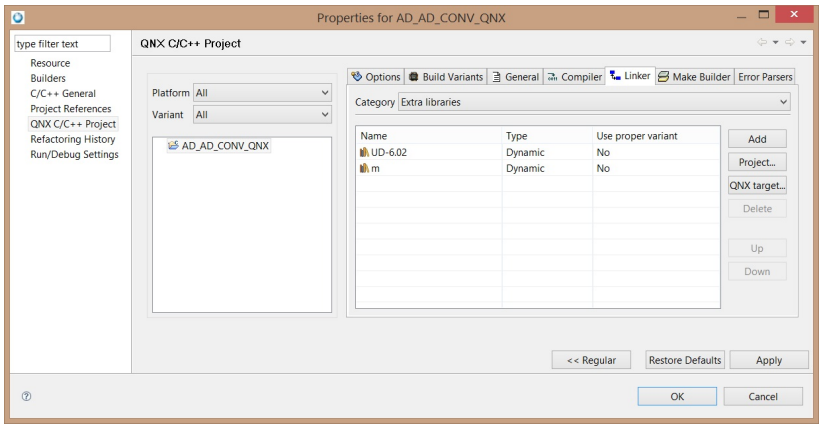

Abbildung 5.3: Einstellung des Projekts zur Einbindung der Bibliotheken

Im letzten Schritt, um das Projekt in Momentics zu konfigurieren, wird dem Compiler das Ziel-System mitgeteilt, auf dem das Programm ausgeführt werden soll. Dies geschieht im Main-Reiter im Feld “Target Options“. Damit das ausführbare Programm auf dem Ziel-System verbleibt, wird im Reiter “Upload“ das Häkchen vom Auswahlfeld “Remove uploaded components after session“ entfernt. Somit ist es möglich, das Programm direkt vom Ziel-Rechner auszuführen. Dies verhindert unerwartete Fehler bei der Ausführung des Programms, da dieses nicht über das Netzwerk ausgeführt wird. Diese beiden Einstellungen sind in 5.1 abgebildet.

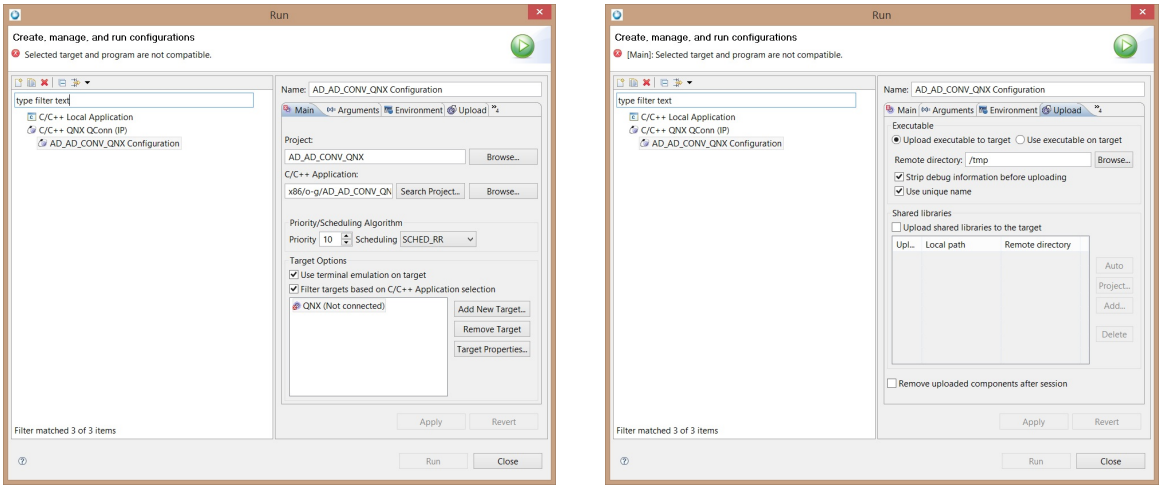

Abbildung 5.4: Projekteinstellung und Zuordnung des Ziel-Systems

Da alle Vorbereitungen durchgeführt sind, kann zu Testzwecken das automatisch generierte Programm von QNX, welches im Listing 5.1 zu sehen ist, vom Host-System kompiliert und ausgeführt werden. Dieses Programm besteht aus einem einfachen printf-Befehl, der auf der Konsole ausgegeben wird. Wird dieses im Konsolenbereich des Entwicklungsprogramm angezeigt, so steht die vorbereitenden Einstellung richtig konfiguriert worden. Um zu überprüfen, ob die ausführbare Datei nach Beendigung der Ausführung über Momentics auf dem eingebetteten System verblieben ist, wird der Inhalt des Ordners im Root-Verzeichnis “/tmp/“ angezeigt. Daraufhin wird das Programm direkt auf dem QNX-Rechner getestet, indem dieses durch einen Doppelklick oder über die Shell mit “./tmp/PROGRAMM“ durchgeführt wird.

Listing 5.1: Automatisch generiertes Programm von QNX

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(int argc, char *argv[]) {
5         printf("Welcome to the QNX Momentics IDE\n");
6         return EXIT_SUCCESS;
7 }
```

5.2 Includes

Nach erfolgreicher Durchführung der Konfiguration der Entwicklungsumgebung wird mit der Umsetzung des Konzepts begonnen. Entscheidend dabei sind zu Beginn die “Includes“, die der Präprozessor nutzt, um allen Quelldateien beim Kompilieren bestimmte Funktionen zu eröffnen, da der Programmiersprache C diese sonst nicht zur Verfügung ständen, vgl. [19, Seite 24 ff]. Die Funktionen stehen in Header-Dateien, die mit dem Befehl “#include“ eingebunden werden. In der C-Programmierung gibt es Standard-Bibliotheken, dessen Inhalt standardisierte Funktionen, wie zum Beispiel der printf()-Befehl aus der Header-Datei “stdio.h“, sind. Des Weiteren können eigene Header-Dateien inkludiert werden, die selbst geschriebene Funktionen beinhalten.

Im Listing 5.2 werde alle inkludierten Header-Datei aufgeführt, die zur Bearbeitung dieses Projekts notwendig sind. Die Header-Datei “stdio.h“ beinhaltet Standard-Funktionen der Ein- und Ausgabe, wie zum Beispiel der Eingabe von Zeichen mit “getc()“ oder der Umgang mit Dateien. “stdlib.h“ wird benötigt, um Speicher für den A/D- und D/A-Puffer dynamisch zu allokieren. Darüber hinaus bietet diese Header-Datei weitere Funktionen im Bereich der Speicherverwaltung, die in diesem Projekt keine Verwendung finden. Der Header “unistd.h“ definiert symbolische Konstanten, die den Standard des Portable Operating System Interfaces (POSIX) festlegen. POSIX dient als definierte Schnittstelle zwischen Anwendungsprogrammen und Betriebssystemen. Verschiedene POSIX-Typen werden ebenfalls definiert, wie zum Beispiel zur Aufnahme der Prozess-Identifikationsnummer eines Threads. Für die Umsetzung der drei verschiedenen Scheduling-Verfahren unter QNX wird der Header “sched.h“ benötigt. Dieser beinhaltet Funktionen zur Konfiguration und Verwaltung des Schedulers. Um Werte in einer Struktur möglichst effizient zu initialisieren, wird dafür die Funktion “memset()“ genutzt. Diese ist in der Header-Datei “string.h“ hinterlegt. Die Datei “time.h“ bietet Zeit-Funktionen an. Das Sporadic-Scheduling nutzt diesen Header für die Funktion “nsec2timespec()“, die eine Zahl in das Zeit-Format überführt. Die Header-Datei “neutrino.h“ liefert QNX-Neutrino-Standards und Funktionen. Zusätzlich inkludiert dieser allgemeine Header, soweit diese noch nicht in der Main-Funktion enthalten sind. Damit das Programm eine parallele Verarbeitung mittels Threads ermöglicht, wird die Datei “pthread.h“ eingebunden. Diese bietet Funktionen zur Verwaltung von Threads an. Zudem ist “PThreads“ ein POSIX-Standard. Damit verbunden sind Funktionen, die zum Beispiel dem Starten und Beenden von Threads dienen. Des Weiteren sind Werkzeuge zur Thread-Synchronisation enthalten. Die letzte inkludierte Header-Datei ist der Treiber “Universal Driver“ von Diamond System, der board-spezifische Funktionen, wie zum Beispiel den Watchdog-Timer, A/D-, D/A und DIO-Operationen anbietet.

Listing 5.2: Inkludierte Header-Dateien

```
1 #include <stdio.h>              // functions for standard I/O
2 #include <stdlib.h>             // functions for memory management
```

```
3 #include <unistd.h>              // POSIX-Standard definition
4 #include <sched.h>               // scheduling functions
5 #include <string.h>              // function for strings
6 #include <sys/time.h>           // time and date functions
7 #include <sys/neutrino.h>       // QNX Neutrino specific functions
8 #include <pthread.h>            // function for thread management
9 #include "dscud.h"              // diamond driver includes
```

5.3 Symbolische Konstanten, globale Variablen und Funktionsdeklarationen

Um den Programmcode lesbar zu halten, werden symbolische Konstanten verwendet, wie im Listing 5.3 dargestellt ist. Diese werden mit dem Befehl “#define“ gesetzt. Für dieses Projekt werden diese Konstanten zum einen für das Sporadic-Scheduling verwendet, das als Parameter sowohl die geringe, als auch die normale Priorität hat. Diese bekommen als Konstanten die Namen LOW_PRIORITY und NORMAL_PRIORITY mit den Werten 10 und 9 übergeben. Zum anderen werden weitere symbolische Konstanten für die A/D- und D/A-Operationen benötigt, die sich aus der Abtastrate (CONV_RATE), FIFO-Größe (FIFO_DEPTH) und der Anzahl der Abtastwerte (NUM_CONV) zusammensetzen. Diese Werte sind notwendig, um die A/D- und D/A-Operationen zu parametrisieren. Die Benutzersteuerung für das Programm wird über digitale Ein- und Ausgänge durchgeführt. Dafür werden alle Zahlen der Eingangspins des C-Registers mit dem Namen “PIN_C#“ ersetzt, wobei das Zeichen “#“ die entsprechende Zahl für den Pin darstellt.

Listing 5.3: Symbolische Konstanten

```
1 // macros define
2 #define NORMAL_PRIORITY 10      // Sporadic Normal-Prio
3 #define LOW_PRIORITY 9          // Sporadic Low-Prio
4 #define CONV_RATE 499           // conversion rate
5 #define FIFO_DEPTH 16           // used FIFO depth
6 #define NUM_CONV 1024           // number of conversion
7 #define PIN_EXIT 7              // exit pin (7)
8 #define PIN_CO 0
```

Zum einfachen Umschalten verschiedener Scheduling-Verfahren werden zusätzlich zwei symbolische Konstanten angelegt. Damit das Programm mit dem gewünschten Scheduling-Verfahren kompiliert wird, muss die dazugehörige Konstante auf “1“ gesetzt werden, da die Einstellung des Schedulings mit einer IF-Struktur gefiltert ist und somit lediglich das gewünschte Verfahren kompiliert wird. Im Listing 5.4 ist sowohl Sporadic als auch Round-Robin mit “0“ ersetzt worden. Somit wird das FIFO-Scheduling kompiliert.

Listing 5.4: Auswahl des Scheduling-Verfahren

```
1 // selection of the scheduling method
2 #define SPORADIC_SET 0
3 #define RR_SET 0
```

Threads eines Prozesses nutzen gemeinsame Ressourcen. Daher werden gemeinsam genutzte Variablen global angelegt. Diese sind in Listing 5.5 dargestellt. Die ersten Deklarationen sind Strukturen und Variablen für den Universal Driver. Diamond Systems bietet neben dem Hercules II EBX-Board noch weitere Boards an. Damit der “Universal Driver“ erkennt, um welches Board es sich handelt, wird in die Variable “dscb“ vom Typ DSCB das aktuell genutzte Board hinterlegt. “dsccb“ ist eine Variable von der Struktur DSCCB, dessen Inhalt die Einstellungen des Boards sind. Zum einen ist dort der Typ und die Nummer des Boards hinterlegt. Zum anderen ist dort die “base-address“ angegeben. Diese Adresse besagt, wo die zu nutzenden Adressen des Registers beginnen, vgl. [17, Seite 105 f]. Dies beschreibt also den Offset der Adressen. Bei dem Hercules II EBX-Board beträgt der Offset 0x240. Weitere Elemente der Struktur sind die Frequenzen der Clocks. Die beiden Variablen werden hauptsächlich zur Initialisierung des Boards für den Befehl “dscInitBoard()“ benötigt. Die nächsten beiden Variablen der Typen DSCADSETTINGS, DSCDASETTINGS werden für die Einstellungen der A/D- und D/A-Operationen benötigt, die ebenso als Strukturen angelegt sind. In der A/D-Struktur werden die Polarität des Signals, die Spannungsbreite (Range) und der Verstärkungsfaktor angegeben. Die restlichen Elemente sind für dieses Projekt nicht relevant. Die D/A-Struktur beinhaltet unter anderem die Polarität des Ausgangssignal. Ebenso wie in der A/D-Struktur werden die restlichen Elemente der Struktur nicht weiter erläutert. Die A/D- und D/A-Operationen benötigen jeweils eine weitere Variable der Struktur DSCAIOINT. Dessen relevanter Inhalt sind die Abtastrate, die Anzahl der Abtastwerte und die Größe des FIFO-Speichers. Des Weiteren ist das Element “cycle“ enthalten, welches angibt, ob es sich um eine einzige A/D- oder D/A-Operation handelt, oder ob es sich um immer wiederholende Operationen handelt. Für das Debugging wird die Variable “errorPar“ verwendet, die im Fehlerfall einer Universal-Driver-Funktion den entsprechenden Fehler ausgibt. Die DSCS-Variablen der A/D- und D/A-Operationen geben den Status der jeweiligen Interrupts an. Diese werden mit dem Befehl “dscGetStatus()“ gefüllt und dienen zur Überprüfung des Interrupts. Inhalt dieser Variable ist der Status des Interrupts, die Anzahl der Abtastwerte aus dem FIFO-Speicher, die gesamte Anzahl der Abtastwerte und die Anzahl der Überfüllung des FIFO-Speichers, vgl. [17, Seite 142]. Da sowohl der A/D-Interrupt als auch der D/A-Interrupt im Programm gleichzeitig laufen, erhält jeder Interrupt seine eigene Variable zur Überprüfung. Die nächsten vier Variablen des Universal Drivers sind vom Typ BYTE. Dieser entspricht dem Typ “unsigned char“. “port_i“ gibt an, welcher Port genutzt werden soll. Um die Richtung der einzelnen Pins zu bestimmen, wird die Variable “config_byte“ verwendet. Diese gibt an, ob es sich um Eingabe- oder Ausgabe-Pins handelt. Für die Steuerung des Programms mit den digitalen Eingängen mittels Schaltern werden die Variablen “exit_prog“

und “C0“ als Flags verwendet, um zum einen das Programm zu beenden und zum anderen die Threads zum Hochzählen zu starten.

Listing 5.5: Deklaration globaler Universal-Driver-Variablen und Strukturen

```
1  // Universal Driver var
2  DSCB dscb;                          // board var
3  DSCCB dsccb;                        // structure containing board settings
4  DSCADSETTINGS adsettings;           // structure for A/D-signal-settings
5  DSCDASETTINGS dasettings;           // structure for D/A-signal-settings
6  DSCAIOINT ad_dscaioint;             // AD structure containing conversion settings
7  DSCAIOINT da_dscaioint;             // DA structure containing conversion settings
8  ERRPARAMS errorPar;                 // structure for returning error messages
9  DSCS da_dscs, ad_dscs;              // used for da and ad interrupts parameters
10 BYTE port_i;                        // port input used for digital I/O
11 BYTE config_byte;                   // DIO configuration byte
12 BYTE exit_prog;                     // flag for exit programm
13 BYTE C0;                            // flags for the c-ports
```

Für die Nutzung des Schedulings der Threads werden Systemvariablen benötigt, die den aufgerufenen Thread anhand einer Nummer identifizieren und die Scheduling-Parameter beinhalten. Des Weiteren werden Hilfsvariablen benötigt. Diese sind in Listing 5.6 abgebildet.
Die Variable “param“ von der Struktur “sched_param“ dient dem Parametrisieren des gewünschten Scheduling-Verfahrens. Diese können mit den dafür vorgesehenen Funktionen gesetzt und abgefragt werden, vgl. [20]. Inhalt dieser Variable ist die vergebene Priorität (sched_priority) und die vom Betriebssystem aktuell genutzte Priorität (sched_curpriority). Die weiteren Elemente der Variable sind Parameter für das Sporadic-Scheduling-Verfahren, die bereits im Grundlagenkapitel beschrieben sind. Die Variable “policy“ repräsentiert das Scheduling-Verfahren anhand einer Zahl. Die Umsetzung ist in der Header-Datei “sched.h“ definiert. Damit die Threads eindeutig identifiziert werden können, bekommen sie beim Aufrufen eine positive Zahl zugewiesen, die in den Variablen “AD“, “DA“ und “DIO“ vom Typ “pthread_t“ abgelegt werden. Um die Ablaufparameter eines Threads zu bestimmen, wird ein Attributobjekt vom Typ “pthread_attr_t“ angelegt. Mit diesem Objekt können dem aufgerufenen Thread die Attribute übergeben werden, um die Priorität und das Scheduling-Verfahren festzulegen.
Die restlichen Variablen sind Hilfsvariablen und dienen zur Unterstützung im Programmablauf. Die Variable “i“ wird vom D/A-Thread für die for-Schleife genutzt. Des Weiteren nutzt der Thread “read_da_b“, um das Auslesen des Puffers mit den Abtastwerten des A/D-Threads zu regulieren. Dabei wird der Puffer wie ein Ringspeicher behandelt und nur eine bestimmte Anzahl von Werten pro Durchlauf in der while-Schleife gelesen und in den eigenen Puffer geschrieben. Die Counter-Threads nutzen die Variable “counter0“ zum Hochzählen. Die Qualität des Wertes dieser Variable ist dabei irrelevant. Sie dient lediglich der Beschäftigung der Threads, damit diese Prozessorzeit verbrauchen, um die Scheduling-Verfahren auf ihre Qualität zu testen. Die letzte Variable “i0“ wird genutzt, um die Anzahl der Counter-Threads

zu zählen. Dieser Wert ist das Qualitätsmerkmal der Scheduling-Verfahren. Denn je mehr Counter-Threads die Prozessorzeit verbrauchen, so dass das System immer noch analoge Werte aufnehmen, umrechnen und wieder ausgeben kann, desto besser ist das genutzte Scheduling-Verfahren.

Listing 5.6: Deklaration globaler System- und Hilfsvaraiblen

```
1  // QNX var
2  struct sched_param param;        // scheduling-structure
3  int policy;                      // scheduling method
4  pthread_t        ad, da, dio;    // thread var
5  pthread_attr_t      attr;        // thread attributes
6  // miscellaneous var
7  int i;                           // counter var
8  int read_da_b;                   // read the FIFO-buffer
9  unsigned long counter0;          // counter for consume processor time
10 int i0;                          // amount of counting threads
```

Der Kern des Programms sind die A/D- und D/A-Funktionen. Diese starten und kontrollieren die A/D- und D/A-Interrupts auf ihre korrekte Ausführung. Beim Starten der beiden Threads wird ihnen die Funktion, in der sie ausgeführt werden sollen, als Parameter mitgegeben. Dies sind die ersten beiden Funktionsdeklarationen im Listing 5.7. Die nächste Deklaration ist die Funktion “dioT()“ zum Abrufen der digitalen Eingänge, die zur Steuerung des Programms dienen. Dies ist eine Funktion für den DIO-Thread. Die letzte Thread-Funktion “tcounter0()“ wird von vielen Threads genutzt, um das System auf ihre Stabilität zu prüfen. Darin wird die Anzahl der Threads gezählt und eine Variable hochgezählt. Die Initialisierungsfunktionen dienen der Übersichtlichkeit des Programmcodes, damit nicht alle Initialisierungen in der Main-Funktion durchgeführt werden. Darin enthalten sind die Initialisierungen für die A/D- und D/A-Interrupts und für den digitalen Eingabeport.

Listing 5.7: Funktionsdeklaration

```
1 // init and thread functions
2 void *adc();
3 void *dac();
4 void *dioT();
5 void *tcounter0();
6 void adInit();
7 void daInit();
8 void dioInit();
```

5.4 Main-Funktion

Zu Beginn der Main-Funktion und somit direkt nach dem Start des Programms werden die Überschrift “AD/DA-Konvertierung in Echtzeit“ und ein Hilfe-Text, damit der Benutzer die Steuerung des Programms kennenlernt, ausgegeben. Danach wird der Eingangsport mit der Variable “port_i“ festgelegt, die die Zahl 2 übergeben bekommt. Diese Variable wird den DIO-Funktionen übergeben und zeigt im Register auf den C-Port. Der Zähler für die Counter-Threads wird auf “0“ gesetzt.

Im Anschluss folgt das Setzen des Scheduling-Verfahrens. Dies ist mit einer IF-Bedingung realisiert. Je nach Auswahl im Define-Bereich, der im Listing 5.4 abgebildet ist, wird das entsprechende Scheduling-Verfahren übernommen. Prinzipiell sollte nur eines der Defines auf “1“ gesetzt werden. Geschieht dies nicht, so wird das Scheduling-Verfahren übernommen, dessen Bedingung als erstes erfüllt ist. Werden beide Defines auf “0“ gesetzt, so wird das FIFO-Scheduling durchgeführt. Das Programm ist somit auf alle drei Scheduling-Verfahren vorbereitet und kann mit geringem Aufwand umgestellt werden.

Bei der Nutzung des Sporadic-Scheduling-Verfahrens wird “SPORADIC_SET“ auf “1“ gesetzt und somit ist die erste IF-Bedingung wahr. Damit beginnt die Konfiguration des Sporadic-Scheduling-Verfahrens, und am Schluss in der IF-Bedingung wird das Verfahren gesetzt. Bei der Konfiguration wird zunächst das Attributobjekt “attr“ mit dem Befehl “pthread_attr_init()“ initialisiert. Somit werden Standardwerte vom System in das Objekt geschrieben. Im Anschluss folgt der Befehl “pthread_attr_setinheritsched()“, der die Vererbung der Attribute für die Threads steuert. In diesem Programm im Sporadic-Scheduling-Verfahren wird festgelegt, dass nur die Attribute, die im Attributobjekt stehen, verwendet werden sollen. Dieses wird mit “PTHREAD_EXPLICIT_SCHED“ eingestellt. Somit bekommen die Threads nicht die Einstellungen des Elternprozesses vererbt. Dies stellt sicher, dass sich die Scheduling-Eigenschaften nicht ändern. Um das Sporadic-Scheduling zu parametrisieren, werden in die Variable “param“ die benötigten Werte eingetragen. Es wird die geringe Priorität festgelegt, damit der Scheduler auf diese wechselt, sobald der Thread die zur Verfügung stehende Zeit überschritten hat. Dies gibt dem Thread die Möglichkeit die Arbeit fortzusetzen, wenn kein weiterer Thread mit höherer Priorität wartet. Der nächste Parameter ist die normale Priorität, mit der jeder Thread gestartet wird. Darauf folgt die Anzahl der Wiederholungen in einer Periode. Diese gibt an, wie oft ein Thread in einer Periode blockiert werden darf, bevor die Priorität herabgesetzt wird. Die beiden letzten Parameter werden mit der Funktion “nsec2timespec()“ aus der Header-Datei “time.h“ beschrieben, damit die Zahl in das Zeitformat übertragen wird, die von diesen Variablen benötigt wird. Die erste Variable "repl_period“ beschreibt die Periode der synchronen Abarbeitung. Nach dieser Zeit werden die Prioritäten der Threads wieder auf die normale Priorität gesetzt. “init_budget“ beschreibt die Zeit, in der die Threads in normaler Priorität den Prozessor nutzen können. Nach Ablauf der Zeit werden die Prioritäten herabgesetzt.

Damit sind alle wichtigen Parameter für das Sporadic-Scheduling gesetzt, und es folgt das Eintragen des Scheduling-Verfahrens “SCHED_SPORADIC“ in das Attributobjekt. Mit dem Befehl “pthread_attr_setschedparam“ werden die Parameter für die Threads übernommen.

Listing 5.8: Einstellung des Sporadic-Scheduling-Verfahrens

```
1  if(SPORADIC_SET) {
2    // initialize the attributes structure
3    pthread_attr_init(&attr);
4
5    // override the default of inheriting policies from the parent
6    pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
7
8    // configure the sporadic scheduling parameter
9    param.sched_ss_low_priority = LOW_PRIORITY;
10   param.sched_priority = NORMAL_PRIORITY;
11   param.sched_ss_max_repl = 3;
12   nsec2timespec(&param.sched_ss_repl_period, 1024000);   // 1024 usec
13   nsec2timespec(&param.sched_ss_init_budget, 128000);    // 128 usec
14
15   // setting the scheduling policy to sporadic
16   pthread_attr_setschedpolicy(&attr, SCHED_SPORADIC);
17   pthread_attr_setschedparam(&attr, &param);
18 }
```

Die Umsetzung des Round-Robin- und FIFO-Verfahrens ist weniger umfangreich, da diese keine Einstellungsmöglichkeiten bieten. Im Listing 5.9 ist die Umsetzung abgebildet. Dabei werden die Parameter vom System übernommen mit dem Befehl “pthread_getschedparam“, das Scheduling-Verfahren in die Variable “policy“ geschrieben und damit wieder als aktuelle Einstellung mit dem Befehl “sched_setscheduler“ in die Variable “param“ eingetragen.

Listing 5.9: Einstellung des Round-Robin- und FIFO-Verfahrens

```
1  //=========================================================================
2  // SCHEDULING PARAMETER main thread ROUND ROBIN
3  //=========================================================================
4  else if(RR_SET) {
5    pthread_getschedparam(0, &policy, &param);
6    policy = SCHED_RR;
7    sched_setscheduler(0, policy, &param);
8  }
9  //=========================================================================
10 // SCHEDULING PARAMETER main thread FIFO
11 //=========================================================================
12 else {
13   pthread_getschedparam(0, &policy, &param);
14   policy = SCHED_FIFO;
```

```
15      sched_setscheduler(0, policy, &param);
16  }
```

Bei der Nutzung des Treibers “Universal-Driver“ von Diamond Systems sind vier Schritte in der Implementierung wichtig. Zum einen müssen der Treiber mit der Funktion “dscInit()“ und das Board mit der Funktion “dscInitBoard()“ initialisiert werden und zum anderen müssen die System-Ressourcen mit den Befehlen “dscFreeBoard()“ und “dscFree()“ wieder freigegeben werden:

1. Treiber Initialisierung mit dscinit()
2. Board Initialisierung mit dscInitBoard()
3. PROGRAMMABLAUF
4. Board-Ressourcenfreigabe mit dscFreeBoard()
5. System-Ressourcenfreigabe mit dscFree()

In der Main-Funktion geschieht die Initialisierung im nächsten Schritt. Dabei wird erst der Treiber initialisiert. Bei der Initialisierung des Boards wird die Variable “dsccb“ benötigt, die vorher definiert werden muss. Diese beinhaltet den Typ des Boards “DSC_HERCEBX“, die Base-Address mit dem Wert 0x240 und das Interrupt-Level mit dem Wert 5. Nach der Initialisierung des Treibers und des Boards, damit der Treiber “Universal Driver“ genutzt werden kann, folgen die Initialisierungsfunktionen “adInit()“, “daInit()“ und “dioInit()“.

Initialisierung des A/D-Interrupts in der Funktion adInit()

Zu Beginn der A/D-Initialisierung werden die Elemente der Variable “adsettings“ mit dem Befehl “memset()“ auf Null gesetzt. Daraufhin folgt das Setzen der Elemente der Variable, wie in Listing 5.10 zu sehen ist. Dabei wird der Spannungsbereich auf 10 V festgelegt, die Polarität auf Bipolar und der Verstärkungsfaktor auf 1 gesetzt. In der Variablen besteht die Möglichkeit, die Genauigkeit der aufgenommenen Werte zu erhöhen, indem “load_cal“ auf TRUE gesetzt wird, vgl. [17, Seite 102]. Dies ist für dieses Projekt nicht erforderlich. Da in diesem Projekt nur ein Kanal für die Abtastung der Spannungswerte genutzt wird, bekommen die Variablen “current_channel“ und “scan_interval“ den Wert 0 zugewiesen. Somit wird nur der Kanal 0 abgetastet. Da alle nötigen Parameter eingestellt sind, werden die Einstellungen mit der Funktion “dscADSetSettings()“ übernommen.

Listing 5.10: Initialisierung der adsettings-Struktur

```
1  memset(&adsettings, 0, sizeof(DSCADSETTINGS));
2
3  adsettings.range = RANGE_10;
4  adsettings.polarity = BIPOLAR;
5  adsettings.gain = GAIN_1;
6  adsettings.load_cal = FALSE;
```

```
7   adsettings.current_channel = 0;
```

Die Parameter für den A/D-Interrupt werden in der Variablen “ad_dscaioint“ gesetzt. Wie in der Variablen “adsettings“ werden auch hier die Elemente durch die Funktion “memset()“ mit dem Wert 0 initialisiert. Im Anschluss folgt die Definition der Elemente, wie das Listing 5.11 zeigt. Die Anzahl der Abtastungen in einer Periode wird auf “NUM_CONV = 1024“ gesetzt. Die Abtastrate beträgt “CONV_RATE = 499“. Somit wird circa alle 2 msek ein Wert abgetastet. Dies ist der maximal einstellbare Wert, wenn der D/A-Interrupt im Programm gleichzeitig durchgeführt wird. Damit der A/D-Interrupt in einer Dauerschleife abtastet, wird das Element “cycle“ auf TRUE gesetzt, vgl. [17, Seite 104]. Ebenso wird das Element “internal_clock“ auf TRUE gesetzt. Das bedeutet, dass die Interrupts mit der internen Clock des Boards generiert werden. Die Elemente “low_channel“ und “high_channel“ werden, wie in der Variablen zuvor, auf 0 gesetzt, da lediglich der Kanal 0 zur Abtastung genutzt wird. Die nächsten beiden Elemente in Zeile 7 und 8 werden auf FALSE gesetzt, da diese für dieses Board “Hercules II EBX“ keine Funktion haben. Damit der A/D-Thread nicht bei jeder Abtastung ein Interrupt auslöst, wird der interne FIFO-Speicher aktiviert und mit der Kapazität von “FIFO_DEPTH = 16“ Abtastwerten eingerichtet. Der “dump_treshold“ bekommt den gleichen Wert zugewiesen. Dieser gibt an, ab welchem Abtastwert ein Interrupt ausgelöst wird und der FIFO-Speicher somit geleert wird. Dabei werden die Werte in den selbst erstellten Puffer des A/D-Threads kopiert, dessen Speicher in der nächsten Zeile mit der Größe der Anzahl der Abtastungen des Elementes “num_conversions“ allokiert wird.

Listing 5.11: Initialisierung der ad_dscaioint-Struktur

```
1   ad_dscaioint.num_conversions = NUM_CONV;
2   ad_dscaioint.conversion_rate = CONV_RATE;
3   ad_dscaioint.cycle = TRUE;
4   ad_dscaioint.internal_clock = TRUE;
5   ad_dscaioint.low_channel = 0;
6   ad_dscaioint.high_channel = 0;
7   ad_dscaioint.external_gate_enable = FALSE;
8   ad_dscaioint.internal_clock_gate = FALSE;
9   ad_dscaioint.fifo_enab = TRUE;
10  ad_dscaioint.fifo_depth = FIFO_DEPTH;
11  ad_dscaioint.dump_threshold = FIFO_DEPTH;
12  // allocate space for sample buffer ad
13  ad_dscaioint.sample_values = (DSCSAMPLE*)malloc( sizeof(DSCSAMPLE) *
14                  ad_dscaioint.num_conversions );
```

Initialisierung des D/A-Interrupts in der Funktion daInit()

Die Initialisierung des D/A-Interrupts hat einen ähnlichen Ablauf wie die Initialisierung des A/D-Interrupts. Der einzige Unterschied besteht in der Variablen “dasettings“. Dies ist eine eigene Struktur für den D/A-Interrupt und beinhaltet lediglich die Polarität, die ebenfalls auf bipolar gesetzt wird, und die nicht benötigte Kalibrierung. Danach werden die Werte mit der Funktion “dscADSetSettings()“ übernommen. Die Variable “da_dscaioint“ ist vom selben Typ wie die Struktur des A/D-Interrupts und beinhaltet die gleichen Parameter. Der D/A-Thread benötigt einen eigenen Puffer, in der dieser die Abtastwerte vom A/D-Puffer übernimmt, da die Werte umgerechnet werden müssen auf Grund der niedrigeren Auflösung von 12 Bit. Der ADC hat hingegen eine Auflösung von 16 Bit. Somit muss der DAC die Werte beim Kopieren um vier Stellen nach rechts verschieben. Dieser Puffer wird ebenfalls mit der gleichen Größe wie der des ADC’s allokiert.

Initialisierung des DIO-Threads in der Funktion dioInit()

In Listing 5.12 ist die Initialisierung der digitalen Steuereinheit mit Schaltern dargestellt, in der die Richtung des digitalen Signals festgelegt wird. Mit dem Wert 0x00 in der Variable “config_byte“ werden alle Ports als Eingangsports festgelegt. Somit sind die Pins des C-Ports ebenfalls mit den Schaltern als Eingang gesetzt, damit die Steuerung durch den Benutzer des Programm durchgeführt werden kann. Die Übernahme der gesetzten Variable erfolgt mit der Funktion “dscDIOSetConfig()“.

Listing 5.12: Initialisierung der DIO-Threads

```
1 void dioInit()
2 {
3   // DIO Setup
4   config_byte = 0x00; // all ports input
5   dscDIOSetConfig(dscb, &config_byte );
6 }
```

Nach der Beendigung der Initialisierung startet der Main-Prozess den A/D-, D/A- und DIO-Thread, wie im Listing 5.13 zu sehen ist. Dafür wird die Funktion “pthread_create()“ genutzt. Als Übergabeparameter bekommt die Funktion zum einen die Adresse von der Identifikationsvariable des jeweiligen Threads, in der eine eindeutige Nummer gespeichert wird. Damit besteht in der Entwicklung eines Programms die Möglichkeit, einen Thread gezielt anzusprechen, um Änderungen in dessen Laufeigenschaft vorzunehmen. Zum anderen wird der Funktion die Adresse des Attributobjektes übergeben, in der die Parameter für die Laufeigenschaft stehen. Damit der Thread erkennt, welche Funktion auszuführen ist, wird im folgenden die Adresse der Thread-Funktion übergeben. Mit dem letzten Parameter kann der Thread einen Wert in seine Startfunktion übergeben bekommen. Da die Thread-Funktionen keine Übergabewerte benötigen, wird dem Thread NULL übergeben, vgl. [21, Seite 47 f].

Listing 5.13: Start der Threads

```
1  //=========================================================================
2  // START THREADS
3  //=========================================================================
4  pthread_create(&ad, &attr, &adc, NULL);
5  pthread_create(&da, &attr, &dac, NULL);
6  pthread_create(&dio, &attr, &dioT, NULL);
```

Nach dem Erstellen der Threads werden diese quasi-parallel durchgeführt, da es sich um ein Einprozessorsystem handelt. Das bedeutet, dass sich die Threads die zur Verfügung stehende Prozessorzeit teilen. Die Zuordnung der Zeit wird durch den Scheduler reguliert. Der Main-Prozess wird im Anschluss durch die Funktion “pthread_join()“ blockiert, bis die erstellten Threads beendet werden. Der Funktion wird die Thread-Identifikationsnummer übergeben, auf der gewartet werden soll, vgl. [21, Seite 57]. Zudem besteht die Möglichkeit, mit dem zweiten Parameter den Wert des Threads zu speichern, den dieser aus seiner Funktion generiert. Dies ist in dieser Thread-Funktion nicht notwendig. Somit wird der Funktion NULL übergeben. Dies ist in Listing 5.14 aufgeführt.

Listing 5.14: Auf Beendigung der Threads warten

```
1  pthread_join(da, NULL);
2  pthread_join(ad, NULL);
3  pthread_join(dio, NULL);
```

Nachdem alle Threads beendet sind, setzt der Main-Prozess seine Arbeit fort mit den Zeilen aus Listing 5.15. Dieser befindet sich nun in der Endphase und bereitet die Beendigung des Programms vor, indem dieser die Systemressourcen und die allokierten Speicher freigibt. Dies geschieht mit der Funktion “free()“ und den beiden “Universal Driver“-Funktionen zur Freigabe der Systemressourcen. Zudem wartet der Main-Thread zum Schluss des Programms auf eine Tastatureingabe mit der Funktion “getchar()“. Die Eingabe wird im Speziellen nicht benötigt. Sie dient lediglich als eine Art Wartefunktion, bis der Benutzer die Informationen auf der Konsole gelesen hat. Daraufhin wird das Programm beendet.

Listing 5.15: Ressourcenfreigabe

```
1  //=========================================================================
2  // CLEANUP
3  //=========================================================================
4  free( ad_dscaioint.sample_values );
5  free( da_dscaioint.sample_values );
6  dscCancelOp(dscb);
7  dscFree();
8  printf("press any key to close...");
9  getchar();
10
11 return 0;
```

5.5 ADC-Funktion

Beim Start des A/D-Threads werden zu Beginn die Elemente der “ad_dscs“-Variablen auf 0 zurückgesetzt. Der Interrupt-Status wird auf “OP_TYPE_INT“ gesetzt, da der A/D-Interrupt im nächsten Schritt gestartet wird, wie im Listung 5.16 zu sehen ist. Beim Start des A/D-Interrupt wird ein eigener System-Thread erstellt, der vom Treiber “Universal Driver“ verwaltet und vom Betriebssystem kontrolliert wird, vgl. [17, Seite 99]. Somit besteht nur die Möglichkeit, die System-Threads mit den Funktionen des Treibers zu verwalten. Die Funktion zum Aufruf des System-Threads “dscADSampleInt()“ bekommt die Variabel “dscb“, in der die Informationen des Boards enthalten sind, und die Variable “ad_dscaioint“, in der die Einstellungen des A/D-Interrupts enthalten sind, übergeben.

Listing 5.16: A/D-Thread-Funktion: Initialisierung und Start des System-Threads

```
1  // values return to zero
2  ad_dscs.transfers = 0;
3  ad_dscs.overflows = 0;
4  ad_dscs.op_type = OP_TYPE_INT;  // interrupt-status
5
6  // start AD Sampling
7  dscADSampleInt( dscb, &ad_dscaioint );
```

Funktionsweise des A/D-System-Threads

Die Funktion des A/D-System-Threads beim Abtasten der Werte ist in mehreren Schritten unterteilt. Der ADC ist ebenfalls im Handbuch des Boards Hercules II EBX [6, Seite 95 ff] beschrieben.

1. Registereintrag des Low-Eingangskanals in BASE+2

Tabelle 5.1: Register BASE+2 A/D Low-Eingangskanal, vgl. [6, Seite 75]

Bit:	7	6	5	4	3	2	1	0
Name:	-	-	-	L4	L3	L2	L1	L0

Im ersten Schritt wird der Low-Eingang der Spannung im Register BASE+2 angegeben. Dieser gibt an, welcher Eingang für das Low-Signal am Board benutzt wird, vgl. [6, Seite 75].

2. Registereintrag des High-Eingangskanals in BASE+3

Tabelle 5.2: Register BASE+3 A/D High-Eingangskanal, vgl. [6, Seite 75]

Bit:	7	6	5	4	3	2	1	0
Name:	-	-	-	H4	H3	H2	H1	H0

Der zweite Schritt ist der Eintrag des High-Einganges im Register BASE+3, vgl. [6, Seite 75].

3. Registereintrag des Spannungsbereiches (Range) in BASE+4

Tabelle 5.3: Register BASE+4 (Write) Eingangsspannungsreichweite, vgl. [6, Seite 75]

Bit:	7	6	5	4	3	2	1	0
Name:	LDAP	-	-	-	-	-	G1	G0

In BASE+4 wird die Range der Spannung angegeben, die der ADC benutzen soll. Dies hat Auswirkung auf die Auflösung des Signals. Je geringer die Range, desto höher ist die Auflösung des Signals, vgl. [6, Seite 75].

4. Warten auf die Übernahme der Registereinträge in BASE+4, Bit 6

Tabelle 5.4: Register BASE+4 (Read) A/D Status, vgl. [6, Seite 76]

Bit:	7	6	5	4	3	2	1	0
Name:	ADBUSY	WAIT	DABUSY	DABU	SEDIFF	ADBU	G1	G0

Das System benötigt eine gewisse Zeit zur Übernahme der Registereinträge. Solange kann noch keine A/D-Umwandlung durchgeführt werden. Das System muss warten, bis die Einträge umgesetzt sind. Dies ist im Bit 6 aus dem Register BASE+4 "WAIT“ ersichtlich. Solange das Bit gleich 1 ist, ist das System noch nicht bereit. Sobald das Bit auf 0 gesetzt wurde, kann die A/D-Umwandlung beginnen, vgl. [6, Seite 76].

5. Starten einer A/D-Umwandlung durch das Setzen des Registers BASE+15, Bit 0

Tabelle 5.5: Register BASE+15 (Write) Kommandos, vgl. [6, Seite 80]

Bit:	7	6	5	4	3	2	1	0
Name:	-	-	FIFORST	DARST	CLRT	CLRD	CLRA	ADSTART

Um eine A/D-Umwandlung zu starten, muss das Bit 0 aus dem Register BASE+15 “ADSTART“ auf 1 gesetzt werden. Dies startet eine A/D Operation, vgl. [6, Seite 80].

6. Warten bis die A/D-Umwandlung fertig ist. Einzusehen im Register BASE+4, Bit 7

Tabelle 5.6: Register BASE+4 (Read) A/D Status, vgl. [6, Seite 76]

Bit:	7	6	5	4	3	2	1	0
Name:	ADBUSY	WAIT	DABUSY	DABU	SEDIFF	ADBU	G1	G0

Während der Durchführung der A/D-Umwandlung wird ein Flag “ADBUSY“ gesetzt. Dieser befindet sich im Register BASE+4 in Bit 7 und gibt an, ob das System noch mit der Abtastung beschäftigt ist. Bei einer 1 muss das System warten. Sobald das Bit auf 0 gesetzt wurde, ist das System mit der Umwandlung fertig, vgl. [6, Seite 76].

7. Abtastwert mit der Auflösung 16 Bit in den FIFO schreiben von BASE+0 und BASE+1

Tabelle 5.7: Register BASE+0 (Read) A/D LSB, vgl. [6, Seite 74]

Bit:	7	6	5	4	3	2	1	0
Name:	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

Tabelle 5.8: Register BASE+1 (Read) A/D MSB, vgl. [6, Seite 75]

Bit:	7	6	5	4	3	2	1	0
Name:	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8

Sobald das System mit der Abtastung fertig ist, wird der Wert aus dem Register BASE+0 und BASE+1 gelesen und in den FIFO geschrieben, vgl. [6, Seite 74, 75].

Nachdem der A/D-Thread den System-Thread zum Abtasten der Spannungswerte gestartet hat, beginnt dieser den A/D-Interrupt in einer While-Schleife auf dessen Funktion zu überprüfen. Der Thread soll so lange in der While-Schleife verbleiben, bis sich der Interrupt selbst beendet, der FIFO überläuft oder der Benutzer durch die digitale Steuereinheit mit dem Schalter C7 das Programm beendet. Wenn der Interrupt endet, setzt der System-Thread das Element “op_type“ aus der Variablen “ad_dscs“ auf “OP_TYPE_NONE“. Der Programm-Code der Schleife ist in Listing 5.17 abgebildet.
Zu Beginn der Schleife wartet der Thread eine Mikrosekunde, um den Prozessor nicht zu überlasten. Daraufhin fragt dieser den Status des A/D-Interrupts mit der Funktion “dscGetStatus()“ ab. Dieser bekommt die Variable “dscb“ mit dem Inhalt des Boards und die Adresse

der Variablen “ad_dscs“, in die der Status des Interrupts geschrieben wird, übergeben.
Wird der System-Thread durch eines der zuvor beschriebenen Bedingung beendet, so springt der A/D-Thread aus der While-Schleife und wird mit der Funktion “pthread_exit()“ beendet. Da der Thread keinen Wert übergeben soll, wird in die Funktion als Parameter eine 0 geschrieben.

Listing 5.17: A/D-Thread-Funktion: While-Scheife zur Überprüfung des Interrupts

```
1  while(ad_dscs.op_type != OP_TYPE_NONE && !ad_dscs.overflows && !exit_prog )
2  {
3    usleep(1);
4    dscGetStatus(dscb, &ad_dscs);
5  }
6
7  pthread_exit(0);
```

5.6 DAC-Funktion

Der D/A-Thread führt die DAC-Funktion aus, dessen Aufgabe es ist, den D/A-Interrupt zu starten, zu überprüfen und die Abtastwerte aus dem Puffer des A/D-Interrupts in seinen eigenen Puffer zu schreiben. Dabei muss der D/A-Thread die Werte jeweils um 4 Bits nach rechts verschieben, da dieser eine geringere Auflösung als der A/D-Interrupt zur Verfügung hat. Der A/D-Interrupt hat eine 16-Bit-Auflösung und der D/A-Interrupt hat lediglich eine 12-Bit-Auflösung. Das bedeutet, dass das Quantisierungsintervall ΔU_q aus der Gleichung 2.3, und somit der kleinstmögliche nächste Schritt der Amplitude bei dem DAC größer ist als bei dem ADC, auf Grund der geringeren Auflösung.

Zu Beginn der Funktion wird das Element “transfers“ der Variablen “da_dscs“ auf 0 zurückgesetzt. Wie auch bei dem A/D-Thread, bekommt das Element “op_type“ der D/A-Struktur den Wert “OP_TYPE_INT“ zugewiesen, damit der Status des D/A-Interrupts für den bevorstehenden Aufruf des D/A-System-Threads aktualisiert wird. Damit der Puffer des DACs nicht bei jedem Lese- und Schreibvorgang den gesamten Puffer des ADCs liest, umrechnet und in den eigenen Puffer schreibt, werden diese Puffer wie ein Ringspeicher behandelt. Das Prinzip wird in Abbildung 5.5 gezeigt.

Somit werden die Abtastwerte blockweise abgearbeitet. Dafür wird die Variable “read_da_b“ benötigt. Diese bekommt zuerst den Wert 0 zugewiesen, damit die Abarbeitung des Puffer-Speichers bei dem ersten Wert beginnt. Im nächsten Schritt erfolgt ein Abarbeitungszyklus, damit der D/A-System-Thread schon vor dem Start aktualisierte Werte in seinem Puffer hat. Dieser Vorgang wird mit einer For-Schleife realisiert. Für diese Schleife wird die Hilfsvariable “i“ verwendet, dessen Aufgabe es ist, hochzuzählen, um die gewünschte Anzahl an Schleifendurchläufen zu realisieren. Dabei wird der Variable “i“ der Wert 0 von “read_da_b“ übergeben. Somit beginnt die For-Schleife bei 0 und der Inhalt der Schleife soll solange ausgeführt werden, wie “i“ kleiner als die Größe eines Blocks “FIFO_DEPTH = 16“ ist.

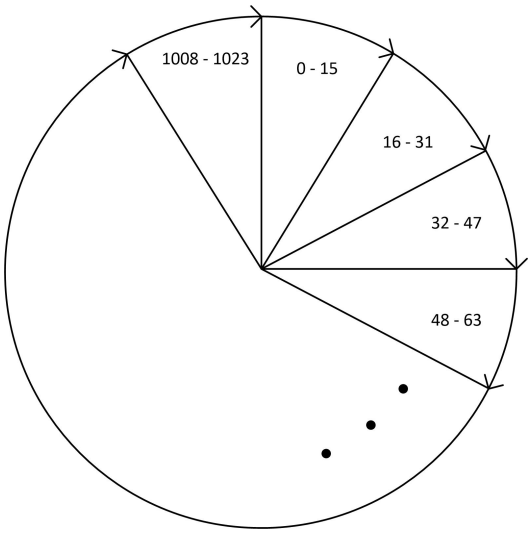

Abbildung 5.5:
Prinzip des Ringspeichers

Bei jedem Durchlauf der For-Schleife werden die Werte des ADC’s gelesen. Diese 16-Bit-Werte müssen auf Grund der geringeren Auflösung von 12 Bit um 4 Bit nach rechts verschoben werden. Damit das Signal weiterhin bipolar ist, wird der Wert 2048 darauf addiert, da dies die Hälfte von $2^{12} = 4096$ ist, wobei die 12 als Exponent die Bitbreite darstellt. Nachdem die 16 Übertragungen der Werte stattgefunden haben, wird diese Variable “read_ad_b“ für den nächsten Block vorbereitet. Dabei wird auf diese Variable “FIFO_DEPTH = 16“ addiert. Die Modulo-Operation mit “NUM_CONV = 1024“ am Ende der Zeile bewirkt, dass der Wert beim Überschreiten der Puffer-Größe wieder bei der Stelle 0 des Speichers beginnt. Nachdem die For-Schleife für einen Block ausgeführt wurde und die Variable “read_da_b“ für den nächsten Block vorbereitet wurde, besitzt diese nun den Wert 16. In Listing 5.18 ist der Code dargestellt.

Listing 5.18: D/A-Thread-Funktion: Initialisierung und Transfer der Abtastwerte

```
1  da_dscs.da_transfers = 0;
2  da_dscs.op_type = OP_TYPE_INT;
3  read_da_b = 0;
4
5  for(i = read_da_b; i < (FIFO_DEPTH+read_da_b); i++) {
6    da_dscaioint.sample_values[i] = (ad_dscaioint.sample_values[i] >> 4) + 2048;
7  }
```

```
8   // modulo-operation for reading buffer in cycle
9   read_da_b = (read_da_b + FIFO_DEPTH) % NUM_CONV;
```

Der nächste Schritt des D/A-Threads ist die Ausführung des D/A-System-Threads mit der Funktion “dscDAConvertScanInt()“, der die Variable “dscb“ mit dem Inhalt des Board-Typs und die Variable “da_dscaioint“ mit dem Inhalt der Interrupt-Parameter übergeben bekommt. Dies ist in Listing 5.20 abgebildet. Dessen Aufgabe ist es, die digitalen Werte aus seinem Puffer in analoge Werte umzuwandeln und auf dem dafür vorgesehenen Kanal auszugeben.

Listing 5.19: D/A-Thread-Funktion: Ausführung des D/A-System-Threads

```
1   // DA Sampling
2   dscDAConvertScanInt( dscb, &da_dscaioint );
```

Funktionsweise des D/A-System-Threads

Um DAC-Operationen durchzuführen, werden Registereinträge von dem D/A-System-Thread getätigt. Somit wird die gesamte Operation einer Ausgabe eines Wertes in mehreren Schritten unterteilt, vgl. [6, Seite 103 ff]:

1. Aufteilen des Wertes in LSB (least significant bit) und MSB (most significant bit)

   Die Umrechnung wird in zwei Schritten aufgeteilt, da das Register eine Breite von 8 Bit hat. Zum einen wird der LSB-Wert durch eine Maskierung herausgefiltert. Dies wird in Gleichung 5.1 dargestellt. Damit sind die ersten 8 niedrig-wertigsten Bits gemeint. Diese werden in das LSB-Register geschrieben. Um die restlichen 4 Bits herauszufiltern, wird der Wert durch $2^8 = 256$ dividiert und die Nachkommastellen fallen durch eine Integer-Umwandlung weg. Diese Berechnung ist in der Gleichung 5.2 zu sehen.

$$\text{LSB} = \text{D/A-Wert} \;\&\; 255 \tag{5.1}$$

$$\text{MSB} = \text{int}\left(\frac{\text{D/A-Wert}}{256}\right) \tag{5.2}$$

2. Ausgangswerte in das Register BASE+6 und BASE+7 schreiben

Tabelle 5.9: Register BASE+6 D/A LSB, vgl. [6, Seite 76]

Bit:	7	6	5	4	3	2	1	0
Name:	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0

Tabelle 5.10: Register BASE+7 D/A MSB, vgl. [6, Seite 77]

Bit:	7	6	5	4	3	2	1	0
Name:	DA15	DA14	DA13	DA12	DA11	DA10	DA9	DA8

Nach der Aufteilung des Wertes in LSB und MSB werden die Werte in das Register BASE+6 für die LSB-Werte und in das Register BASE+7 für die MSB-Werte geschrieben, damit das Signal auf den Ausgangskanal ausgegeben werden kann, vg. [6, Seite 76, 77].

3. Ausgangskanal im Register BASE+5, Bit 1, 2 bestimmen

Tabelle 5.11: Register BASE+5 D/A-Kanal, vgl. [6, Seite 76]

Bit:	7	6	5	4	3	2	1	0
Name:	SU	-	-	-	-	-	DACH1	DACH0

Der A/D-System-Thread kennt aus der Variablen “da_dscaioint“ den Ausgangskanal für das analoge Signal. Dieser wird in das Register BASE+5 in den ersten beiden Bits eingetragen, vgl. [6, Seite 76].

4. Auf die Ausgabe des Wertes warten im Register BASE+4, Bit 5 “DABUSY“

Tabelle 5.12: Register BASE+4 (Read) A/D Status, vgl. [6, Seite 76]

Bit:	7	6	5	4	3	2	1	0
Name:	ADBUSY	WAIT	DABUSY	DABU	SEDIFF	ADBU	G1	G0

Mit dem Registereintrag für den ausgewählten Ausgangskanal wird die D/A-Operation ausgeführt. Damit wird DABUSY aus dem Register BASE+4 auf 1 gesetzt. Das bedeutet, dass die D/A-Operation ausgeführt wird. In dieser Zeit werden keine weiteren Ausgangswerte in das Register geschrieben. Erst bei Beendigung der D/A-Operation wird DABUSY wieder auf 0 gesetzt und die nächste Operation kann durchgeführt werden, vgl. [6, Seite 76].

Der D/A-System-Thread führt die D/A-Operation in einer Endlosschleife durch. Um diese Operationen zu überprüfen, folgt im weiteren Verlauf des Codes, welcher im Listing 5.20 zu sehen ist, eine While-Schleife, dessen Inhalt der A/D-While-Schleife ähnelt. Ebenso sind die Abbruchbedingungen der While-Schleife gleich. Die Schleife wird erst dann verlassen, wenn sich der Interrupt-Status des Elementes “op_type“ auf OP_TYPE_NONE ändert, der FIFO-Speicher überlauft oder das Programm durch den Benutzer mit dem Schalter C7 an der

digitalen Schnittstelle beendet wird. Um den Prozessor nicht zu überlasten, wartet der D/A-Thread eine Mikrosekunde und stellt im Anschluss eine Statusabfrage für den D/A-System-Thread mit der Funktion “dscGetStatus()“. Der Unterschied zur A/D-While-Schleife ist, dass der D/A-Thread die Werte aus dem Puffer in seinen eigenen Puffer kopiert und vorher an seine Auflösung anhand einer Bitverschiebung anpasst. Dieser Vorgang wird mit einer For-Schleife realisiert, die bereits erklärt wurde, da der D/A-Thread an dieser Stelle des Codes schon einen Block in seinen Puffer geschrieben hat. Der Unterschied ist, dass die Variable “read_ad_b“ nun den Wert 16 hat und der Vorgang somit im zweiten Block beginnt. Solange die Bedingungen der While-Schleife eingehalten werden, führt der Thread diese Operationen der Übertragung der Werte in einem Ringspeicher aus.

Wird die While-Schleife verlassen, so beendet sich der Thread mit der Funktion “pthread_exit“, dessen Übergabeparameter 0 ist, da der Thread keine Werte bei der Beendigung übergibt.

Listing 5.20: D/A-Thread-Funktion: While-Schleife und Beendigung des D/A-System-Threads

```
1  while(da_dscs.op_type != OP_TYPE_NONE && !ad_dscs.overflows && !exit_prog)
2  {
3    usleep(1);
4    dscGetStatus(dscb, &da_dscs);
5    for(i = read_da_b; i < (FIFO_DEPTH+read_da_b); i++) {
6        da_dscaioint.sample_values[i] = (ad_dscaioint.sample_values[i] >> 4)
7                                        + 2048;
8    }
9    // modulo-operation for reading buffer in cycle
10   read_da_b = (read_da_b + FIFO_DEPTH) % NUM_CONV;
11 }
12
13 pthread_exit(0);
```

5.7 DIO-Funktion

Der DIO-Thread wird in der dioT-Funktion quasi-parallel ausgeführt. Dabei beginnt der Thread eine Variable zu deklarieren, in der die Identifikationsnummer der Threads gespeichert wird, die die Prozessorzeit durch eine Counter-Operation verbrauchen sollen. Dies ist in Listing 5.21 zu sehen.

Listing 5.21: DIO-Thread-Funktion: Deklaration der Thread-Identifikationsvariable

```
1  // additional threads for consuming processor time
2  pthread_t tc0;
```

Daraufhin folgt eine While-Schleife, in der die digitalen Eingänge des Ports C gelesen werden und in die die dafür vorgesehenen Variablen gespeichert werden. Dabei werden lediglich die Pins DIO_C0 und DIO_C7 gelesen und deren Wert gespeichert. Der Abbruch der While-Schleife erfolgt, wenn der FIFO-Speicher überläuft oder der Benutzer das Programm mit dem Schalter C7 beendet. Zu Beginn der While-Schleife wartet dieser Thread eine Mikrosekunde mit der Funktion “usleep(1)“, um den Prozessor so wenig wie möglich zu belasten. Danach werden die Pins DIO_C0 und DIO_C7 mit der Funktion “dscDIOInputBit()“ gelesen und in den Variablen “C0“ und “exit_prog“ gespeichert. Diese Funktion ist in dem Treiber “Universal Driver“ enthalten. Die Übergabeparameter dieser Funktion sind zum einen die Informationen des Boards und zum anderen wird der Funktion mitgegeben, welcher Port gelesen werden soll. Dieser befindet sich im Register BASE+18, welcher in Tabelle 5.13 zu sehen ist. Weitere Parameter sind die Angabe des Pins, von dem gelesen werden soll, und die Variable, in der der Wert gespeichert wird.

Tabelle 5.13: Digitale Ports C im Register BASE+18, vgl. [6, Seite 81]

Bit:	7	6	5	4	3	2	1	0
Name:	DIOC7	DIOC6	DIOC5	DIOC4	DIOC3	DIOC2	DIOC1	DIOC0

Bei Betätigung des Schalters C0 wird in die Variable “C0“ der Wert 1 geschrieben und die folgende If-Bedingung ist erfüllt, so dass eine Ausgabe “start“ auf der Konsole erfolgt. Dies signalisiert dem Benutzer, dass die Schalterbetätigung erfolgreich vom Programm registriert wurde. Daraufhin wird in der If-Abfrage eine weitere While-Schleife durchgeführt, und die Counter-Threads werden nacheinander alle 100 msek gestartet. Dies erfolgt mit der Funktion “pthread_create()“ und bekommt als Parameter die Thread-Identifikations-Variable, das Attributobjekt und die Thread-Funktion übergeben. Da bei den Threads keine Übergabe einer Variable erfolgt, wird als letzter Parameter NULL übergeben. Die While-Schleife wird erst beim Überlaufen des FIFO-Speichers abgebrochen. Solange werden Counter-Threads erstellt, um das System zu belasten. Erst bei einer gewissen Anzahl an Counter-Threads kann der Interrupt des A/D-Threads nicht mehr rechtzeitig die Werte in den FIFO speichern und der FIFO läuft über. Dies ist in Listing 5.22 zu sehen.

Listing 5.22: DIO-Thread-Funktion: Digitale Eingänge lesen und Counter-Threads erstellen

```
1  while(!ad_dscs.overflows && !exit_prog)
2  {
3    usleep(1);
4    dscDIOInputBit( dscb, port_i, PIN_EXIT, &exit_prog );
5    dscDIOInputBit( dscb, port_i, PIN_C0, &C0 );
6
7    if(C0) {
8      printf("start\n\n");
9      while(!ad_dscs.overflows) {
```

```
10          pthread_create(&tc0, &attr, &tcounter0, NULL);
11          dscSleep(100);
12        }
13      }
14    }
```

Wenn die Anzahl der Counter-Threads erreicht ist, so dass der FIFO übergelaufen ist, verlässt der DIO-Thread beide While-Schleifen und gibt zum Schluss die Anzahl der Counter-Threads “i0“ und den Wert der hochgezählten Variable “counter0“ auf der Konsole aus. Danach beendet der DIO-Thread sich selbst mit der Funktion “pthread_exit(0)“. Dies ist in Listing 5.23 abgebildet.

Listing 5.23: DIO-Thread-Funktion: Ausgabe der erreichten Anzahl der Counter-Threads

```
1   // print the reached amount of counter-threads and counter-value
2   printf("Anzahl␣Counter-Threads:␣%d␣und␣Counter:␣%ld\n\n", i0, counter0);
3
4   pthread_exit(0);
```

5.8 Counter-Funktion

Die Counter-Funktion, die von den Counter-Threads ausgeführt wird, beginnt mit einem Inkrement der Variable “i0“. Diese wird um einen Zähler erhöht, sobald ein Counter-Thread die Funktion betritt. Diese Variable wird genutzt, um die Anzahl der Counter-Threads zu zählen, damit das System und dessen Scheduling-Verfahren auf Grund dieser Variable auf Stabilität geprüft wird. Die Anzahl gibt eine Aussage über die Qualität des Scheduling-Verfahrens. Je mehr Counter-Threads die Variable “counter0“ quasi-parallel hochzählen, desto besser kann der Scheduler mit dem laufenden Verfahren die Prozessorzeiten aufteilen und die zeitlichen Bedingungen einhalten. Als nächstes folgt im Code eine While-Schleife, die solange ausgeführt wird, bis der FIFO-Speicher des ADC’s überläuft. In der Schleife warten die Counter-Threads eine Mikrosekunde, bevor die Variable “counter0“ inkrementiert wird. Die Inkrement-Operation ist in den meisten CPU-Architekturen keine atomare Operation, die in einem einzigen Maschinen-Befehl ausgeführt wird. Somit handelt es sich hier um einen kritischen Bereich. Denn die Operation unterliegt der so genannten Race Condition, vgl. [19, Seite 652]. Das bedeutet, dass der zeitliche Verlauf entscheidend für das Resultat der Variable ist, wenn mindestens zwei Threads gleichzeitig auf diese zugreifen. Im Grunde ist es in dieser Situation erforderlich, dass die Threads synchronisiert werden. Dieses kann bei der kleinen Operation mit einem Mutex umgesetzt werden, damit der Wert der Variable nacheinander hochgezählt wird. Da das Resultat dieser Variable in diesem Projekt nicht relevant ist, weil diese nur als Beschäftigungsmaßnahme für die Counter-Threads existiert, entfällt

die Synchronisation. Wenn der FIFO des ADC's überläuft, verlassen die Counter-Threads die While-Schleife und beenden sich mit der Funktion "pthread_exit()" selbst. Diese Counter-Funktion ist in Listing 5.24 abgebildet.

Listing 5.24: Counter-Thread-Funktion

```
1  void *tcounter0()
2  {
3    i0++;
4    while(!ad_dscs.overflows)
5    {
6      usleep(1);
7      counter0++;
8    }
9
10   pthread_exit(0);
11 }
```

6 Qualitätsprüfung der Scheduling-Verfahren

Die Scheduling-Verfahren im Echtzeitbetriebssystem QNX haben unterschiedliche Vor- und Nachteile. Das FIFO-Scheduling ist das einfachste Verfahren, welches sich ohne großen Aufwand implementieren lässt. Zudem wird weniger Prozessorzeit für den Kontextwechsel benötigt, der vom Dispatcher durchgeführt wird, da jeder Prozess seine Abarbeitungszeit verbraucht und im Anschluss den Prozessor für den nächsten Prozess in der Warteschlange freigibt. Das Prinzip des Round-Robin-Verfahren ist ähnlich. Der Unterschied ist, dass dieses Verfahren eine Zeit vorgibt, in der der Prozess seine Aufgabe durchführen kann. Wenn die Zeit verbraucht ist, muss dieser den Prozessor für den nächsten Prozess freigeben und reiht sich am Ende der Warteschlange ein. Diese Zeit wird auch “timeslice“ genannt. Das bedeutet, dass der Dispatcher mehr Prozessorzeit für den Kontextwechsel in Anspruch nimmt. Das aufwendigste Verfahren ist das Sporadic-Scheduling. Hier werden zusätzlich Veränderungen der Priorität während der Laufzeit vorgenommen. Daraus resultiert, dass der Dispatcher viel Zeit in Anspruch nimmt, um die Abarbeitungszeit zu steuern. Dennoch kann dadurch die Abarbeitungszeit effektiver organisiert werden, so dass die Echtzeitfähigkeit möglichst eingehalten werden kann.

Um die Echtzeitfähigkeit der verschiedenen Scheduling-Verfahren zu prüfen, wird eine Anfangsbedingung definiert. Dies stellt sicher, dass bei jedem Durchlauf der Prüfung die gleichen Bedingungen im System bestehen, damit das Ergebnis der Prüfung nicht durch eine leicht veränderte Umgebung verfälscht wird und so die Aussagekraft der Qualität möglichst hoch ist. Die Prüfung besteht aus einem angepassten Quellcode, so dass das Programm von einem Shell-Skript 1000 mal nacheinander ausgeführt wird, um das Ergebnis der Anzahl der Counter-Threads in eine Auswertungsdatei “log.txt“ zu schreiben. Dies erfordert mehrere Stunden, so dass der Testdurchlauf des Shell-Skriptes nachts durchgeführt wird. Ebenfalls wird dadurch sichergestellt, dass das System durch keine weiteren Anwendungen gestört wird. Das Ergebnis wird mit Matlab ausgewertet. Dabei werden die Werte aller drei Scheduling-Verfahren als Normalverteilung dargestellt, so dass daraus die Qualität der Verfahren ersichtlich wird.

6.1 Anfangsbedinung

Jeder Testdurchlauf benötigt die gleiche Anfangsbedingung, um den Test nicht zu verfälschen. Vor der Ausführung des Shell-Skriptes, wird das QNX-System neu gestartet. Das gewährleistet einen definierten Anfangszustand, so dass keine unnötigen Prozesse im Hintergrund laufen und der Arbeitsspeicher nicht zu sehr in Gebrauch ist. Nach dem Neustart werden außer der Shell und dem Shell-Skript keine weiteren Programme geöffnet. Da der Test minimal 15 Stunden dauert, um das Programm mit allen drei Scheduling-Verfahren jeweils 1000 mal durchzuführen, wird dieser am Ende eines Tages gestartet, so dass der Test während der Nacht läuft. Dies gewährleistet, dass keine weiteren Anwendungen auf dem System durchgeführt werden, um den Testdurchlauf nicht zu gefährden.

Da das Shell-Skript das Programm mehrfach ausführt, muss gewährleistet sein, dass das Programm ohne Unterbrechungen durchgeführt wird und nur das Ergebnis der Anzahl der Counter-Threads ausgegeben wird, um diese in eine Auswertungsdatei zu schreiben. Das bedeutet, dass das Programm entsprechend geändert werden muss. Zum einen werden alle unnötigen Zeilen zur Konsolenausgabe entfernt. Zum anderen müssen die Counter-Threads nach einer gewissen Zeit ohne eine Eingabe eines Benutzers nacheinander erstellt werden. Nach dem Start des ADC's und des DAC's erfolgt der Start des DIO-Threads, der in der DIO-Funktion ausgeführt wird. Dabei wird zu Beginn, wie in der normalen Version des Programms, die Thread-Identifikations-Variable angelegt. Danach wartet der Thread drei Sekunden, bevor die Counter-Threads alle 100 msek gestartet werden. Dies wird mit der Funktion "dscSleep(3000)" realisiert. Durch das Warten von drei Sekunden wird gewährleistet, dass der Puffer des ADC's mindestens einmal komplett gefüllt wird. Dies stellt sicher, dass das System stabil läuft. Die angepasste Version der DIO-Funktion ist in Listing 6.1 abgebildet.

Listing 6.1: Veränderte Version für den Test der DIO-Funktion

```
1  void *dioT()
2  {
3    // additional threads for consuming processor time
4    pthread_t tc0;
5
6    dscSleep(3000);    // wait 3 sec
7    while(!ad_dscs.overflows)
8    {
9      pthread_create(&tc0, &attr, &tcounter0, NULL);
10     dscSleep(100);   // wait 100 msec
11   }
12   printf("%d", i0); // print the reached amount of counter-threads
13
14   pthread_exit(0);
15 }
```

6.2 Shell-Skript

Das Shell-Skript ist eine Datei in Unix-Systemen, in der die Kommandos, die in einer Shell ausgeführt werden können, in einer gewissen Reihenfolge stehen, vgl. [22, Seite 248 ff]. Der Vorteil ist, dass somit viele Kommandos mit der Ausführung des Shell-Skriptes durchgeführt werden, ohne dass ein Benutzer die Kommandos nacheinander in die Shell eingeben muss. Um das Shell-Skript ausführen zu können, müssen zuvor die erforderlichen Ausführungsrechte dieser Datei vergeben werden. Dies wird mit dem Befehl “chmod +x“ realisiert. Somit lässt sich die Datei mit “./DATEINAME“ ausführen.

Das Shell-Script, welches zum Testen der Scheduling-Verfahren angewendet werden soll, wird auf dem QNX-Ziel-System geschrieben und mit den Zugriffsrechten ausführbar gemacht. Dabei beginnt das Skript mit der Zeile “#!/bin/sh“. Damit wird das Skript mit dem Interpreter “Bourne-Shell“ ausgeführt, vgl. [23, Seite 23]. Dies ist der Interpreter der Shell-Umgebung. Darauf folgt eine Konsolenausgabe zur Angabe des Shell-Skript-Titels. Der Datei-Pfad der Log-Datei mit den einzelnen Testergebnissen wird in die Variable “LOG“ geschrieben. Dies hat den Vorteil, dass der Pfad beim Schreiben des Skriptes nicht immer angegeben werden muss und dass die Variable nur einmal geändert werden muss, wenn der Pfad sich ändert. Die folgende Zeile erstellt eine leere Datei namens “log.txt“. Dies wird mit dem Befehl “echo“ realisiert. Das Argument “-n“ gibt an, dass der Echo-Befehl nach dessen Ausgabe keinen Zeilenumbruch durchführt. Durch die “>“-Operation wird die Ausgabe in eine Datei geleitet, dessen Inhalt in “$LOG“ steht. Dies ist die Variable, die den Pfad der Log-Datei gespeichert hat. Der Anfang des Shell-Skriptes ist in Listing 6.2 aufgeführt.

Listing 6.2: Erster Teil des Shell-Skriptes zum Test der Scheduling-Verfahren

```
1 #!/bin/sh
2
3 echo "*** test program for several scheduler ***"
4 echo " "
5 LOG=/tmp/log.txt
6
7 # empty file
8 echo -n > $LOG
```

Nach den oben genannten Vorbereitungen des Shell-Skriptes für die Testumgebung folgen die Aufrufe des Programms mit den unterschiedlichen Scheduling-Verfahren. Damit für den Benutzer des Shell-Skriptes deutlich wird, wo das Skript sich in der Abarbeitung des Codes befindet, wird zu Beginn für jedes Scheduling-Verfahren ein Text auf die Konsole ausgegeben. Dieser gibt an, dass das Testverfahren für das jeweilige Scheduling durchgeführt wird. Daraufhin folgt ein Zeitstempel, der in die Auswertungsdatei “log.txt“ geschrieben wird. In der Auswertung der Datei wird dadurch deutlich, wann das jeweilige Testverfahren begonnen hat. Dies wird mit dem Befehl “echo -n ‘date‘ $\gg$ $LOG“ realisiert. Das Argument “-n“ verhindert

den Zeilenumbruch am Ende dieses Befehls, der von dem Echo üblicherweise durchgeführt wird. Der Befehl “date“ ist der Zeitstempel des Unix-Systems. Damit “date“ als Befehl vom Echo-Befehl erkannt wird, müssen um diesen Befehl einfache Anführungszeichen gesetzt werden. Das Zeichen “≫“ leitet die Ausgabe in eine Datei. Dabei wird diese als Zeichenkette angehängt und nicht, wie bei dem Zeichen “>“, an den Anfang der Datei geschrieben. Im weiteren Verlauf des Skriptes wird der Name des Scheduling-Verfahrens in die Log-Datei geschrieben. Danach wird eine Variable “i“ als Laufindex generiert. Diese bekommt den Wert “1“ zugewiesen. Im Anschluss folgt eine While-Schleife, die solange durchlaufen wird, wie “i“ kleiner oder gleich 1000 ist. Der Inhalt dieser Schleife ist der Aufruf des Programms. Das Ergebnis wird als Zeichenkette an die Log-Datei gehängt. Am Ende der Schleife wird die Variable “i“ um den Wert 1 erhöht. Somit wird das Programm 1000 mal aufgerufen. Damit der Benutzer weiß, wann der Test des jeweiligen Verfahrens beendet ist, erfolgt am Schluss eine Konsolenausgabe mit der Zeichenkette “finish“. Die Umsetzung wird im Listing 6.3 gezeigt.

Listing 6.3: Shell-Skript: Test des Sporadic-Verfahrens

```
1  echo -n "SPORADIC test is starting ... "
2  echo -n `date` >> $LOG
3  echo -n  ": " >> $LOG
4  echo "SPORADIC TEST" >> $LOG
5  i=1
6  while [ $i -le 1000 ]
7  do
8     ./AD_AD_CONV_QNX_SPORADIC_TEST >> $LOG
9    echo -n ", " >> $LOG
10    let i=$i+1
11 done
12 echo "finish"
13 echo " "
```

Nach der Beendigung des Sporadic-Verfahrens folgt der Round-Robin-Test und der FIFO-Test. Dieser Ablauf des Skriptes ist identisch mit dem des Sporadic-Tests. Der einzige Unterschied ist, dass das entsprechende Programm aufgerufen wird und die Ausgaben an den Namen des laufenden Scheduling-Tests angepasst werden.

Wenn alle Scheduling-Tests beendet sind, wird ganz zum Schluss ein Zeitstempel in die Datei geschrieben. Somit kann überprüft werden, wie lange der Testdurchlauf gedauert hat.

6.3 Auswertung

Die Ergebnisse der einzelnen Testdurchläufe aller Scheduling-Verfahren wird mit Matlab ausgewertet. Dabei werden die Ergebnisse des jeweiligen Verfahrens als Normalverteilung dargestellt. Dies stellt die Anzahl der Counter-Threads in Bezug auf die Auftrittswahrscheinlichkeit dar.

Zu Beginn werden die 1000 Werte in einen Vektor geschrieben, wie in Listing 6.4 abgebildet ist. Danach wird ein weiterer Vektor erstellt, der alle ganzen Zahlen von 0 bis 999 beinhaltet. Um die Auswertung in der Normalverteilung plotten zu können, wird zunächst der Mittelwert aller Ergebnisse der Scheduling-Verfahren und danach die Standardabweichung gebildet. Diese Ergebnisse werden zur Berechnung der Normalverteilung benötigt, die im Anschluss folgt. Zum Schluss werden die Ergebnisse der Berechnung als Plot dargestellt und von 0 bis 250 begrenzt.

Listing 6.4: Matlab-Skript zur Auswertung des Scheduling-Tests

```
% mit 1000 Werten Versuch 1
spo = [149, 153, 145, 148, 140, 144, ..., 161, 89, 136, 148, 152];
rr = [137, 109, 144, 89, 137, 97, ..., 100, 80, 126, 128, 149];
fi = [128, 144, 144, 144, 152, 135, ..., 98, 101, 119, 113, 143];

b = 0:1:999;

% Mittelwert
MittelSp = mean(spo);
MittelRr = mean(rr);
MittelFi = mean(fi);

% Standardabweichung
StdSp = std(spo);
StdRr = std(rr);
StdFi = std(fi);

% Normalverteilung
pdfNorm_spo = normpdf(b,MittelSp,StdSp);
pdfNorm_rr = normpdf(b,MittelRr,StdRr);
pdfNorm_fi = normpdf(b,MittelFi,StdFi);

% Plot
plot(b,pdfNorm_spo,b,pdfNorm_rr,b,pdfNorm_fi);
legend('Sporadic', 'Round Robin', 'FIFO');
xlim([0 250]);
```

6.4 Ergebnis

Das Ergebnis zeigt die Normalverteilung aller Scheduling-Verfahren. Dieser Test wurde zweimal durchgeführt, um das Resultat der Qualität zu festigen. Dies ist in den Abbildungen 6.1 und 6.2 zu sehen. Wie erwartet, zeigt das Sporadic-Scheduling in beiden Tests eine höhere Qualität, da das Maximum bei der höheren Anzahl der Counter-Threads ist. Dennoch war auffällig, dass im zweiten Test der Mittelwert und die Standardabweichung deutlich geringer sind als im ersten Test. Das hat zur Folge, dass der Plot des zweiten Tests schmaler ist und sich weiter links befindet.

Im zweiten Test ist das System nicht so belastbar, wie es im ersten Test war, was möglicherweise daran lag, dass das System im zweiten Test im Hintergrund mehr Prozesse ausgeführt hat. Insgesamt bedeutet das für dieses Echtzeitsystem, dass das Sporadic-Verfahren mehr Counter-Threads verarbeiten kann, bevor der FIFO-Speicher des ADC's überläuft. Das Round-Robin-Verfahren liegt qualitativ zwischen dem FIFO- und dem Sporadic-Verfahren. Das FIFO-Verfahren zeigt die geringste Qualität. Insgesamt liegen die Qualitäten nicht weit voneinander entfernt. Die Maxima unterscheiden sich bei dem ersten Test in 6 Counter-Threads. Bei dem anderen Test ist der Unterschied noch geringer. Allerdings ist das Resultat in beiden Tests gleichbleibend. Für dieses Echtzeitsystem ist das Sporadic-Scheduling-Verfahren am besten geeignet. Bei diesem Scheduling-Verfahren kann der Scheduler die Prozessorzeit am besten nutzen, so dass die Bedingungen der Echtzeit am effektivsten eingehalten werden können. Die Berechnungen der Mittelwerte und der Standardabweichungen sind in den Tabellen 6.1 und 6.2 zu sehen.

Tabelle 6.1: Ergebnis des ersten Tests

Scheduling-Verfahren	**Mittelwert**	**Standardabweichung**
Sporadic	122,396	22,78
Round-Robin	120,235	22,81
FIFO	117,193	22,67

Tabelle 6.2: Ergebnis des zweiten Tests

Scheduling-Verfahren	**Mittelwert**	**Standardabweichung**
Sporadic	107,390	18,94
Round-Robin	105,651	17,03
FIFO	104,977	16,74

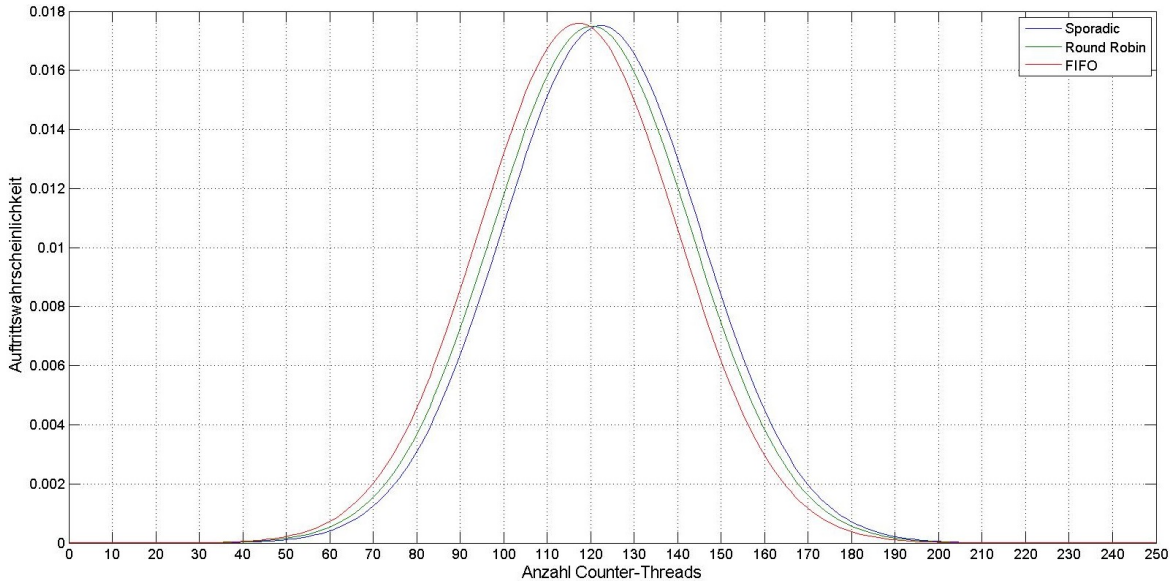

Abbildung 6.1: Test 1 der Scheduling-Verfahren im System

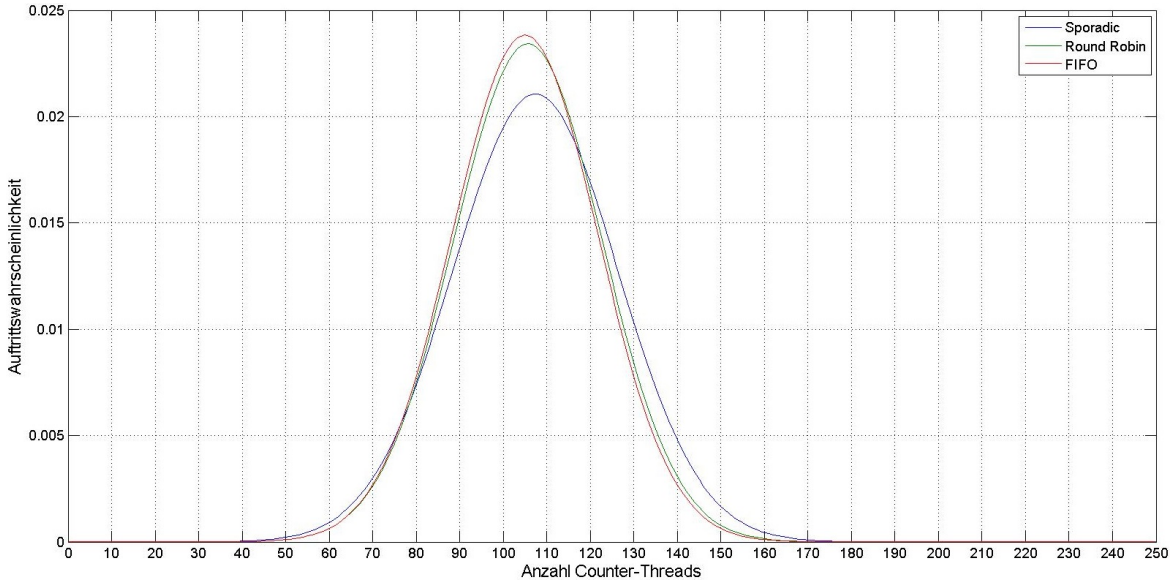

Abbildung 6.2: Test 2 der Scheduling-Verfahren im System

7 Versuchbeschreibung

Die Versuchsbeschreibung für den vierten Labortermin dient als Leitfaden für ein besseres Verständnis in Bezug auf Echtzeitsysteme für die Studierenden des Masterstudiengangs Automatisierung. Die praktische Umsetzung eines Projekts festigt die theoretischen Grundlagen. Des Weiteren werden allgemeine Grundlagen der Betriebssysteme thematisiert. Das Design und das Format der Versuchsbeschreibung sind an die vorherigen Beschreibungen von Herrn Professor Wenck von der HAW Hamburg angelehnt.

4. PRAKTIKUM
Echtzeitsysteme

In diesem Labor soll ein Echtzeitsystem entwickelt werden, das analoge Eingangssignale in digitale Werte umwandelt. Diese Aufgabe wird von einem ADC übernommen. Die digitalen Werte sollen im Anschluss erneut in ein analoges Signal überführt und ausgegeben werden. Diese Aufgabe übernimmt ein DAC. Diese beiden Prozesse stellen den Kern des Echtzeitsystems dar. Da es sich bei dem Zielrechner um ein Einprozessorsystem handelt, werden die Threads quasi-parallel ausgeführt. Dies bedeutet, dass sich die Threads die Zeit des Prozessors teilen müssen. Die Aufgabe der Zuteilung der Zeit wird durch den Scheduler übernommen. QNX bietet drei verschiedene Scheduling-Verfahren (FIFO, Round-Robin, Sporadic). Diese Verfahren haben unterschiedliche Funktionsweisen, Qualitäten und entsprechend Vor- und Nachteile. Daher muss das geeignete Scheduling-Verfahren für das Projekt gefunden werden, welches die Einhaltung der Echtzeit-Bedingungen gewährt.

Hinweis: Für jede Aufgabe wird unter Momentics ein neues Projekt angelegt.

VORBEREITUNG: Aufbau des Systems

1. Verbinden Sie den Funktionsgenerator mit dem eingebetteten System. Dabei achten Sie bitte darauf, dass die ersten analogen Eingangsbuchsen genutzt werden.
2. Zur Überprüfung verbinden Sie den Funktionsgenerator zusätzlich mit einem Kanal des Oszilloskops.
3. Für das Ausgangssignal des Systems verbinden Sie den ersten analogen Ausgang vom Board mit dem Oszilloskop.

4. Achten Sie darauf, dass beim Funktionsgenerator wegen des Boards maximal $\pm 10V$ eingestellt werden darf.

AUFGABE 1: Bereitstellen der Kernfunktionen ADC und DAC

Gegeben ist der Programmcode “ADDACONVQNX.c“, die Bibliothek “libUD-6.02.a“ und der Header “dscud.h“ des Treibers “Universal Driver“ für das Board Hercules II EBX von Diamond Systems. Diese Dateien finden Sie im Emil-Portal im Bereich der Vorlesung “Betriebssysteme und Echtzeitprogrammierung“.

1. Erstellen Sie ein Projekt unter Momentics und fügen Sie den Programmcode aus der Datei “ADDACONVQNX.c“ ein.
2. Fügen Sie die Bibliothek “libUD-6.02.a“ des Treibers und die Mathe-Bibliothek “libm.a“ ein. Die Möglichkeit, diese einzubinden, finden Sie in den Einstellungen des Projekts auf der Seite “QNX C/C++ Project“ in dem Reiter “Linker“.
3. Kopieren Sie die Header-Datei “dscud.h“ in Ihr Projekt, damit die Funktionen des Treibers genutzt werden können.
4. Stellen Sie sicher, dass der Prozess “qconn“ auf dem QNX-Rechner läuft.
5. Binden Sie den QNX-Rechner in Momentics ein. Klicken Sie hierfür auf den Pfeil des Debugger-Symbols, um das Fenster “Open Debug Dialog...“ zu öffnen. Nun müssen Sie unter “C/C++ QNX QConn (IP)“ ein Objekt erstellen, in dem Sie den Zielrechner und die Applikation “/PFAD/DATEINAME_g“ einbinden. Im Reiter “Upload“ entfernen Sie den Haken für das Feld “Remove uploaded components after session“. Damit wird das kompilierte Programm auf dem Zielrechner nicht gelöscht.
6. Überlegen Sie, wie Sie mit dem Programmcode die einzelnen Scheduling-Verfahren auswählen können und kompilieren Sie das Programm auf dem Zielrechner mit allen Scheduling-Verfahren, so dass sowohl das eingehende Signal als auch das umgewandelte, ausgehende Signal auf dem Oszilloskop zu sehen sind. Benennen Sie die Programme entsprechend des Scheduling-Verfahrens auf dem Zielrechner um.
7. Beschreiben Sie die Funktionsweise der drei Scheduling-Verfahren (FIFO, Round-Robin, Sporadic), die das Echtzeitbetriebssystem QNX anbietet.
8. Der ADC tastet mit einer Frequenz von 499 Hz ab. Wie hoch darf die maximale Signalfrequenz sein, damit das Abtasttheorem nicht verletzt wird?
9. Wird eine Schaltereingabe am Steuerelement durch Interrupts oder durch das Polling registriert? Erklären Sie warum.

AUFGABE 2: Automatisches Ablaufen des Programms

In dieser Aufgabe soll das Programm so umgeschrieben werden, dass keine Schaltereingabe des Benutzers erforderlich ist, um die Erstellung der Counter-Threads zu starten, damit das System belastet wird. Zudem soll nur die Ausgabe der Anzahl der Counter-Threads auf die Konsole erfolgen. Zur Umsetzung gelten folgende Bedingungen:

- Nur die Anzahl der erstellten Counter-Threads soll ausgegeben werden.
- Der DIO-Thread soll nach dem Start 3 Sekunden warten, damit der Puffer des ADC's mindestens einmal komplett gefüllt ist.
- Danach soll ohne Betätigung des Schalters die Erstellung der Counter-Threads beginnen.

Das Umschreiben des Programmcodes erfolgt in mehreren Schritten:

1. Entfernen Sie in der Main-Funktion den Anfangstext und den Endtext. Zusätzlich entfernen Sie den "getchar()"-Befehl.
2. Fügen Sie in der DIO-Funktion einen Befehl ein, der den Thread direkt nach dessen Start 3 Sekunden warten lässt.
3. Danach soll der DIO-Thread ohne Betätigung des Schalters C0 die Erstellung der Counter-Threads beginnen. Diesbezüglich wird die While-Schleife in der DIO-Funktion so verändert, dass die Erstellung der Threads nach den 3 Sekunden direkt beginnt.
4. Nach der While-Schleife soll die nur die Anzahl der erstellten Counter-Threads ausgegeben werden.
5. Kompilieren Sie das Programm mit allen drei Scheduling-Verfahren und benennen Sie die Programme im Zielrechner um.
6. Führen Sie die Programme auf dem Zielrechner aus. Dabei muss zunächst das Ein- und Ausgangssignal auf dem Oszilloskop zu sehen sein. Nach einer gewissen Zeit muss sich das Programm selbst beenden und die Anzahl der erstellten Counter-Threads ausgeben.

AUFGABE 3: Test der Scheduling-Verfahren anhand eines Shell-Skriptes

Da QNX ein unixoides Betriebssystem ist, können Shell-Kommandos verwendet werden. Diese können ebenfalls automatisch anhand eines Shell-Skriptes ausgeführt werden. Um eine Aussage über die Qualität der Scheduling-Verfahren zu machen, wird ein Shell-Skript verwendet, das die kompilierten Programme aus Aufgabe 2 mehrfach aufruft. Dabei speichert dieses Skript die empfangenen Werte aus dem Programm in eine Auswertungsdatei "log.txt". Das Shell-Skript ist ebenfalls in Emil enthalten. Die Auswertungsdatei kann im

Anschluss des Tests genutzt werden, um aus jedem Scheduling-Verfahren eine Normalverteilung zu berechnen und als Plot anzuzeigen. Daraus können die Scheduling-Verfahren bewertet werden.

1. Kopieren Sie das Shell-Skript “testing.sh“ auf den Ziel-Rechner.
2. Vergeben Sie mit Hilfe der Shell und dem Befehl “chmod +x DATEI“ die entsprechenden Rechte, die zur Ausführung des Skriptes benötigt werden.
3. Ändern Sie das Shell-Skript derart um, dass die Programme mit den jeweiligen Scheduling-Verfahren 10 mal aufgerufen werden.
4. Starten Sie den Zielrechner neu, damit ein definierter Anfangszustand der Testumgebung geschaffen wird.
5. Starten Sie das Shell-Skript.
6. Kopieren Sie sich die Auswertungsdatei auf dem Host-System, um diese mit Hilfe von Matlab auszuwerten. Dabei lassen Sie sich die Normalverteilung berechnen und plotten.
7. Bewerten Sie das Ergebnis des Plots. Welches Scheduling-Verfahren ist für dieses System am besten geeignet?

8 Fazit / Zusammenfassung und Ausblick

Im letzten Kapitel wird eine zusammengefasste Darstellung zur Umsetzung der Arbeit beschrieben. Dies beinhaltet, wie die einzelnen Systemkomponenten eingesetzt wurden, damit das Echtzeitsystem die ADC- und die DAC-Aufgabe durchführt. Dabei wurde eine Testumgebung implementiert, um die unterschiedlichen Scheduling-Verfahren auf dessen Qualität zu prüfen. Im Nachfolgenden Unterpunkt “Ausblick“ werden Ansatzpunkte beschrieben, die im Anschluss an diese Bachelor-Arbeit weiter bearbeitet werden können. Zusätzlich wird erläutert, wozu das System in Stande ist, wenn bestimmte Themengebiete der Arbeit weiterentwickelt werden.

8.1 Zusammenfassung

Das Echtzeitsystem mit dem Betriebssystem QNX wurde zur Realisierung von Analog-Digital-Umsetzungen und von Digital-Analog-Umsetzungen entwickelt. Dabei wurden die drei unterschiedlichen Scheduling-Verfahren genutzt, um die Zuteilung der Prozessorzeit für die Threads zu organisieren. Diese Scheduling-Verfahren haben unterschiedliche Qualitäten, die im Verlauf der Arbeit durch eine entwickelte Testumgebung geprüft wurden. Für diese Aufgabe wurde das Board “Hercules II EBX“ von Diamond Systems verwendet. Dieses hat die Möglichkeit, analoge und digitale Ein- und Ausgangssignale zu verarbeiten. Im Speziellen wurde darüber das analoge Spannungssignal eingespeist. Der ADC wandelt dieses Signal in ein digitales Signal um. Während der ADC mit der Abtastung des analogen Signals beschäftig ist, läuft quasi-parallel der DAC, der das digitale Signal in ein analoges Signal umwandelt und dieses über den Ausgang ausgibt. Für den quasi-parallelen Ablauf wurden verschiedene Scheduling-Verfahren verwendet, um die Benutzung der CPU zeitlich zu koordinieren. Um die Scheduling-Verfahren auf dessen Qualität zu prüfen, wurde das Programm jeweils mit allen drei Verfahren kompiliert. Ferner wurde ein Thread entwickelt, der alle 100 msek einen weiteren Thread erstellt, die in Addition eine Variable hochzählen, um das System zu belasten. Durch dieses Vorgehen ist es möglich, die Scheduling-Verfahren auf dessen Effektivität und Qualität zu testen. Je mehr Counter-Threads die Variable hochzählen, um das System zu belasten, desto besser ist das laufende Scheduling-Verfahren.

Für die Testumgebung wurde ein Shell-Skript geschrieben, das die Aufgabe hat, das Programm mit jedem Scheduling-Verfahren jeweils 1000 mal auszuführen, um die Anzahl der Counter-Threads, die beim Zusammenbruch des Systems der Konsole übergeben werden, in eine Auswertungsdatei zu schreiben. Diese Datei wurde im Anschluss verwendet, um die Werte in Matlab zu übernehmen, damit dieses Programm die Normalverteilung berechnet. Mit dieser Verteilung wurde eine Aussage über die Qualität des Scheduling-Verfahrens getroffen.

8.2 Ausblick

Der ADC hat im Zusammenspiel mit dem DAC eine maximale Abtastrate von 499 Hz, da der A/D-Interrupt bei höherer Abtastfrequenz den FIFO-Speicher für die Abtastwerte nicht rechtzeitig kopieren und leeren kann. Das bedeutet, dass das Signal alle 2 msek abgetastet wird. Durch das Abtasttheorem ist das Eingangssignal auf maximal 249 Hz begrenzt. Dies ist kein hoher Wert und entspricht in etwa eine Periodenzeit von 4 msek. Für die Reaktionszeit einiger Echtzeitsysteme ist die Signalfrequenz noch ausreichend. Jedoch gibt es viele sicherheitsrelevante Systeme, die eine höhere Reaktionszeit benötigen. Als Beispiel ist in der Airbag-Steuerung eine Reaktionszeit im Millisekunden-Bereich erforderlich. In dieser Zeit wertet das Airbag-System die Messwerte der Sensoren aus, damit gewährleistet wird, dass das menschliche Leben möglichst gut geschützt wird. Daher kann an dieser Stelle angesetzt werden, um den ADC und den DAC für weitaus höhere Werte stabil einzustellen. Technisch ist für den ADC eine maximale Abtastrate von 10 MHz zu erreichen. Jedoch kann das Verhalten der System-Threads, die von dem Treiber “Universal Driver“ gestartet werden, nicht eingeschätzt werden, da der Entwickler darauf keinen Einfluss hat. Der Treiber wird dem Entwickler als Bibliothek zur Verfügung gestellt. Dies erschwert die Anpassung und die Verbesserung des Systems. Möglicherweise müssen für die Optimierung des ADC’s und des DAC’s die Systeme selbst entwickelt werden. Das bedeutet, dass der Treiber für diese Aufgaben nicht benutzt werden kann. Somit hat der Entwickler deutlich mehr Möglichkeiten, in das System einzugreifen und Optimierungen vorzunehmen, da zu jeder Zeit genau bekannt ist, was das System macht.

8.3 Hinweis zum Anhang

Der Anhang zur Arbeit befindet sich auf CD und ist einzusehen bei den Prüfern Professor Wenck und Professor Hess der Hochschule für Angewandte Wissenschaften Hamburg. Darin enthalten sind der Programmcode, das Shell-Skript, das Matlab-Skript und die genutzte Literatur, die in digitaler Form zur Verfügung stand.

Literaturverzeichnis

[1] WOERN, Heinz ; BRINKSCHULTE, Uwe: *Echtzeitsysteme*. Springer, 2005. – ISBN 3–540–20588–3

[2] BB10QNX: *Microkernel: Für Embedded Systeme die bessere Wahl*. https://bb10qnx.de/2014/07/microkernel-fuer-embedded-systeme-die-bessere-wahl/

[3] HÖHER, Peter A.: *Grundlagen der digitalen Informationsübertragung*. Springer, 2013. – ISBN 978–3–8348–1784–6

[4] MICHEEL, Hans J.: *Systemtheorie*. Vorlesungsskript, 2010

[5] MANDL, Peter: *Grundkurs Betriebssysteme*. Springer, 2014. – ISBN 978–3–658–06217–0

[6] DIAMOND SYSTEMS CORPORATION (Hrsg.): *Hercules II-EBX User Manual*. 1.04. 1255 Terra Bella Ave., Mountain View, CA 94043: Diamond Systems Corporation, 2010

[7] BENRA, Juliane T. ; HALANG, Wolfgang A.: *Software-Entwicklung für Echtzeitsysteme*. Springer, 2009. – ISBN 978–3–642–01595–3

[8] SOFTWARE-KOMPETENZ: *Begriffsdefinition: Echtzeitfaehigkeit, Rechtzeitigkeit, Gleichzeitigkeit, Jitter, Determinismus*. http://www.software-kompetenz.de/servlet/is/28612/?print=true

[9] PROKOP, Michael: *QNX*. http://michael-prokop.at/computer/qnx.html

[10] WIKIPEDIA: *QNX*. https://de.wikipedia.org/wiki/QNX. Version: 2015

[11] WIKIPEDIA: *Interprozesskommunikation*. https://de.wikipedia.org/wiki/Interprozesskommunikation. Version: 2015

[12] QNX SOFTWARE SYSTEMS LIMITED (Hrsg.): *QNX Neutrino RTOS System Architecture*. 1. 1001 Farrar Road, Ottawa, Ontario, K2K 0B3, Canada: QNX Software Systems Limited, 2014

[13] WIKIPEDIA: *Sporadic Scheduling*. https://de.wikipedia.org/wiki/Sporadic_Scheduling. Version: 2014

[14] TIEPIE ENGINEERING (Hrsg.): *Handyscope HS3 User manual*. 2.13. Koperslagersstraat 37, 8601 WL Sneek, Niederlande: TiePie Engineering, 2015

[15] HEWLETT PACKARD COMPANY (Hrsg.): *Operating and Service Manual Model 3310A 3310B*. 1. P.O. Box 301, Loveland, Colorado 80537 U.S.A.: Hewlett Packard Company, 1973

[16] TIEPIE ENGINEERING (Hrsg.): *Multi Channel Software User Manual*. 1.05. Koperslagersstraat 37, 8601 WL Sneek, Niederlande: TiePie Engineering, 2014

[17] DIAMOND SYSTEMS CORPORATION (Hrsg.): *Universal Driver*. 6.01. 1255 Terra Bella Ave., Mountain View, CA 94043 USA: Diamond Systems Corporation, 2009

[18] QNX SOFTWARE SYSTEMS LIMITED (Hrsg.): *Ten Steps to Your First QNX Program*. 6.4. 1001 Farrar Road, Ottawa, Ontario, K2K 0B3, Canada: QNX Software Systems Limited, 2008

[19] GOLL, Joachim ; DAUSMANN, Manfred: *C als erste Programmiersprache*. Springer, 2014. – ISBN 978–3–8348–1858–4

[20] QNX: *sched param*. http://www.qnx.com/developers/docs/6.4.0/neutrino/lib_ref/s/sched_param.html. Version: 2015

[21] QNX SOFTWARE SYSTEMS LIMITED (Hrsg.): *Get Programming with the QNX Neutrino RTOS*. 6.6. 1001 Farrar Road, Ottawa, Ontario, K2K 0B3, Canada: QNX Software Systems Limited, 2014

[22] WOLFINGER, Christine ; UNGER, Herwig: *Keine Angst vor Linux Unix*. Springer, 2013. – ISBN 978–3–642–32078–1

[23] EHSES, Erich ; KÖHLER, Lutz ; RIEMER, Petra ; STENZEL, Horst ; VICTOR, Frank: *Systemprogrammierung in UNIX Linux*. Teubner, 2012. – ISBN 978–3–8348–1418–0

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 14. Januar 2016	
Ort, Datum	Unterschrift