Bachelorarbeit

Bent Mucha

Erstellung eines Mess-Systems zur Latenzzeitmessung
von Bluetooth SPP Verbindungen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Bent Mucha

Erstellung eines Mess-Systems zur Latenzzeitmessung von Bluetooth SPP Verbindungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof Dr. rer. nat. Thomas Canzler

Abgegeben am 05. August 2008

Bent Mucha

Thema der Bachelorarbeit

Erstellung eines Mess-Systems zur Latenzzeitmessung von Bluetooth SPP Verbindungen

Stichworte

Bluetooth, Serial Port Profile (SPP), Latenz, serielle Verbindung, Windows, Funktechnologie.

Kurzzusammenfassung

Diese Arbeit behandelt die Erstellung eines geeigneten Latenzzeitenmess-Systems für Bluetooth-SPP-Verbindungen unter Einsatz eines auf Windows basierenden PC. Eine SPP-Verbindung ersetzt in er Regel eine kabelgebundene, serielle Verbindung, wodurch der Einsatz in der Steuerung von Maschinen, Industrieanlagen, etc. erzielt werden kann. Das System soll hierbei den Aufwand für den Aufbau eines Labors umgehen, das Gelingen soll anhand von geeigneten Tests bewiesen werden.

Bent Mucha

Title of the paper

Development and construction of a measuring system to measure the latency of a Bluetooth SPP connection

Keywords

Bluetooth, Serial Port Profile (SPP), latency, serial connection, Windows, radio communication.

Abstract

This report deals about the construction of an easy to use System for measuring the latencies of a Bluetooth SPP connection under the usage of a windowsbased Computer. The SPP connection usually replaces a serial connection over a cable. In this case it is usable for the communication between robots, industrial machines, etc. The system is intended for minimizing the complexity of building up a laboratory. Some tests shall proof the correct functionality afterwards.

INHALTSVERZEICHNIS

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

EINLEITUNG

1.1 Motivation

Am Anfang war die Idee: Wireless.

Den Aufwand der Verkabelung diverser Geräte im Alltag von privaten und industriellen Anlagen galt es zu minimieren: Computermäuse, Tastaturen, PDAs, elektrische Steuerungsanlagen, etc. mussten noch in den 90er Jahren per Kabel verbunden werden. Eine Alternative musste her und wurde sehr bald gefunden, die Idee der kabellosen Verbindung. Während zu Beginn des neuen Jahrtausends das Thema Bluetooth noch milde belächelt wurde, es war nicht klar ob sich diese Technik durchsetzen würde, ist diese Technik heute nicht mehr wegzudenken. Es existiert kaum ein Handy, dass sich nicht über Bluetooth mit dem PDA synchronisieren kann oder gleichzeitig eine drahtlose Verbindung zur Freisprecheinrichtung aufzubauen vermag. Bereits im Jahr 2005 wurden pro Woche ca. 5 Millionen Geräte mit Bluetoothfunktion verkauft, Tendenz steigend (2003: 1 Mio./Woche) [GOLEM]. Die Industrie fand schnell Gefallen an der Funktechnik, war sie doch durch höher werdende Stückzahlen sehr günstig und durch das lizenzfreie Frequenzband (siehe Kapitel 2.2.1) ohne Folgekosten.

Es ergaben sich zudem neue Möglichkeiten in der Steuerung und Realisierung von Robotern und autonomen Transportfahrzeugen. Vorher wurde jede Verbindung per Kabel realisiert, dies führte leider oft dazu, dass Verbindungen durch häufige Bewegungen (Roboterarme) verschlissen, dadurch anfälliger auf Kabelbrüche wurden und regelmäßig getauscht werden mussten [ABB]. Problematisch waren auch autonome Fahrzeuge (automated guided vehicles), wie sie beispielsweise in der Druckerei vom Axel-Springer-Verlag zum Transport der tonnenschweren Papierrollen eingesetzt werden [AGV], da die Steuerung von autonomen Fahrzeugen durch eine Leitstelle bisher nur per Kabel oder Infrarot erfolgen konnte. Die Fahrzeuge in der Druckerei navigierten derzeit auf festen, vorher definierten Routen, welche farblich auf dem Hallenboden markiert wurden. Unter den Markierungen befanden sich nun Kabel, welche ein Magnetfeld induzierten (Floor Wire [ROCLA]). Die Übertragung der Anweisungen durch die Leitstelle konnte nur an bestimmten Punkten (i. d. R. Endpunkte) via Infrarot übertragen werden, da hier immer der direkte Sichtkontakt zwischen Sender und Empfänger gewährleistet werden musste. Der Einsatz von Kabeln war ferner problematisch, da jederzeit Stolpergefahr für die Mitarbeiter

EINLEITUNG

bestand und auch Kabelbruch nicht auszuschließen war. Schließlich wird der Wirkungsbereich stark eingeschränkt. Funktechnologie war Anfang der 90er Jahre nicht einsetzbar, es gab kein Verfahren zur Vermeidung von Störungen. Die Firma Ericsson wurde erst 1994 beauftragt, eine auf Funkwellen basierende Lösung mit dem Ziel des Kabelersatzes zu finden [WIKI]; WLAN wurde erst gegen 1997 für die kommerzielle Nutzung erschlossen [AW]. In der Theorie stellt dieser Sachverhalt (Nutzung des Floor Wire) kein Problem dar, jedoch konnte das Fahrzeug einen Fehler/Störung auf der Strecke nur schwierig an die Leitstelle melden, da es keine permanenten Datenverbindungen gab.

Durch den Einsatz von Funktechnologien können diese Probleme weitestgehend gelöst werden. Bereits heute existiert ein System der Firma SENA Technologies Inc. [SENA], welches im Bereich Facility Management eine fertige Funkinfrastruktur mit Bluetooth für autonome Fahrzeuge bereitstellt. Theoretisch ließe sich mittels Bluetooth eine Vielzahl von kabelgebundenen Verbindungen drahtlos realisieren. Es gäbe, neben den oben aufgeführten Problemen, auch weniger Schwierigkeiten mit herstellerspezifischen Steckverbindungen. Auch Gewicht und Kosten für Kabelbäume in jeder Art von Fahrzeugen ließen sich minimieren. Jedoch stellt sich für den versierten Entwicklungsingenieur eine Frage: Wenn man das Kabel in Steuerungsanlagen, Roboterarmen, etc. ersetzt, wie ändert sich dann die Übertragungsgeschwindigkeit? Der Kupferdraht ist immer schneller als die Übertragung per Funktechnologie im Bezug auf den physikalischen Aufbau. Mit welcher Verzögerung aber muss für die Übertragung gerechnet werden? Ist diese immer gleich, oder hängt dies vom Gerät ab? Spielt die Senderichtung eine Rolle? Der Entwickler kann anhand von ermittelten Verzögerungszeiten schnell erkennen, ob die von ihm einzusetzen beabsichtigte Hardware den von ihm gestellten Anforderungen genügt. Beispielsweise ist die durchschnittliche Reaktionszeit einer Maschine, etc. innerhalb einer bestimmten Zeit vorausgesetzt, der Entwickler weiß aber vorher nicht, ob diese Voraussetzung überhaupt machbar ist. Ein Vertreter kann einen Endkunden leicht von seinem Produkt überzeugen das richtige zu sein, wenn er vor Ort und schnell den IST-Zustand vorweist, der positiv beworben werden soll. Ein Entwickler, zuständig für die Programmierung der Firmware, kann schneller testen, ob sich Veränderungen in der Software positiv oder negativ auf das Latenzverhalten ausgewirkt haben. Die ortsunabhängige Nutzbarkeit lässt viele Einsatzmöglichkeiten zu, nahezu jede Umgebung ist

EINLEITUNG

nutzbar, speziell solche, wo nur geringe Platzverhältnisse vorherrschen, welche sich in ständiger Bewegung befinden (Automobil, Flugzeug, Schiff), oder wo der Aufbau eines weitaus komplexeren Systems nicht so leicht möglich wäre (Industrieanlage). Diese und noch weitere Einsatzgebiete sind für das Mess-System denkbar, es gilt also nun ein System zu erstellen, welches die genannten Kriterien erfüllen kann.

1.2 Zielsetzung

Ziel ist es, ein Mess-System zu erstellen, womit man schnell und mit einfachen Mitteln die Latenzzeit einer Funkverbindung zwischen zwei Bluetoothmodulen messen kann. Jenes System soll auf einer leicht zu beschaffenden Hardware realisiert werden. Da in Firmen meist PCs mit Windowssystem anzutreffen sind, ist hier Windows – basierend auf NT – die primäre Wahl. Zudem soll das System auf einem handelsüblichen Laptop funktionieren, ein Entwickler kann damit ortsunabhängig auch bei einem Kunden, in einem Fahrzeug, etc. ein Mess-System zu Demonstrations- oder sonstigen Zwecken aufbauen. In Verbindung mit Entwicklungssystemen aber auch generell in der Industrie wird für die Signalverarbeitung meist eine serielle Schnittstelle verwendet, daher soll das zu erstellende System auch hierauf aufbauen. Die bisherige Methode ein solches System aufzubauen war in der Regel mit mehr Aufwand und mit umfangreicherer Hardware verbunden, die zu bedienen nicht unbedingt einfach oder leicht zu verstehen war. Diese Arbeit bezieht sich bezüglich der Kommandos auf den Gerätepool der Firma Stollmann E+V GmbH.

1.3 Gliederung der Arbeit

In Kapitel 2 wird zunächst die für diese Arbeit verwendete Hardware und Software des Gesamtsystems vorgestellt. Die Funktionsweise der Bluetoothprotokolle soll erläutert werden, wie auch das Windowssystem in seinen Eigenschaften mit der seriellen Ein-/Ausgabe und den bereitgestellten Funktionen der WindowsAPI (Application Program Interface).

EINLEITUNG

Kapitel 3 befasst sich mit der Analyse des zu Erstellenden Systems. Was muss es können, welche Randbedingungen/Umgebungen sind notwendig oder vorhanden? Gibt es bereits jetzt Einschränkungen in der späteren Funktionalität?

Kapitel 4 zeigt das Design der nötigen Software und Alternativen bezüglich der Erstellung. Im Anschluss werden die Grundfunktionen anhand der verwendeten Prozeduren erläutert. Diese werden einzeln inklusive der Übergabeparameter und Rückgabewerte vorgestellt.

Kapitel 5 prüft die Funktionsfähigkeit der Software anhand einiger Tests wie sie auch später durchgeführt werden sollen.

Kapitel 6 vergleicht die Softwaremessung mit einer weitaus präziseren Hardwaremessung und zeigt eventuelle Ungenauigkeiten. Hierzu werden einige Testläufe mit entsprechender Messhardware nachvollzogen und direkt gegenübergestellt. Schließlich sind einige Praxisanwendungen der Bluetoothwelt aufgeführt.

Kapitel 7 gibt ein abschließendes Fazit und einen Ausblick auf mögliche Weiterentwicklungen der Mess-Software.

GRUNDLAGEN

2.1 Windows NT [WNT, WIKI]

Die erste Frage die geklärt werden musste war, ob das angedachte Mess-System auf einem auf Windows basierenden PC überhaupt praktikabel ist – welche Latenzzeiten ergeben sich bereits für die Ausgabe eines einzelnen Zeichens und welche Abfolgen müssen durchlaufen werden, bis das Signal am ComPort (serielle Schnittstelle) anliegt?

Der Anwender hat in der Regel keinen direkten Zugriff auf die Hardwareebene des Gesamtsystems. Dies stellt ein recht hohes Maß an Sicherheit für das System dar, soll doch der Nutzer weitestgehend von den Grundlagen des Betriebssystems getrennt und damit eklatante Fehlfunktionen/Schäden am System durch unbeabsichtigte, wie auch beabsichtigte Manipulationen verhindert werden. Dies ist aber wiederum problematisch für die Verarbeitung von Anfragen an eine Hardwarekomponente. Zum besseren Verständnis wird zunächst der grundlegende Aufbau eines Windowssystems basierend auf NT erläutert.

Der Aufbau eines Betriebssystems kann zunächst unterschiedlich strukturiert sein. Man unterscheidet hier zwischen den monolithischen Systemen, geschichteten Systemen und Client/Server Modellen. Betriebssysteme der Windows Familie, welche auf der Architektur von NT basieren (Windows 2000, XP, Vista), vereinen die letzen zwei Modelle. Zum besseren Verständnis der Unterschiede werden alle drei Modelle aufgeführt.

2.1.1 Monolithisches System (Windows 9x, MS-DOS)

In diesem System sind die Prozeduren so angeordnet, dass es jeder Prozedur erlaubt wird, eine Andere aufzurufen. Dies führt in der Regel zu schnellerem und kleinerem Programmcode, birgt jedoch den Nachteil, dass Änderungen in einer Prozedur Fehler in anderen Prozeduren erzeugen, oder aufdecken können. Zudem sind Änderungen an den einzelnen Prozeduren nicht so schnell machbar, da hier der gesamte Kernel neu erzeugt (kompiliert) werden muss, wie es beispielsweise bei den Anfängen der Linuxsysteme nötig war.

GRUNDLAGEN

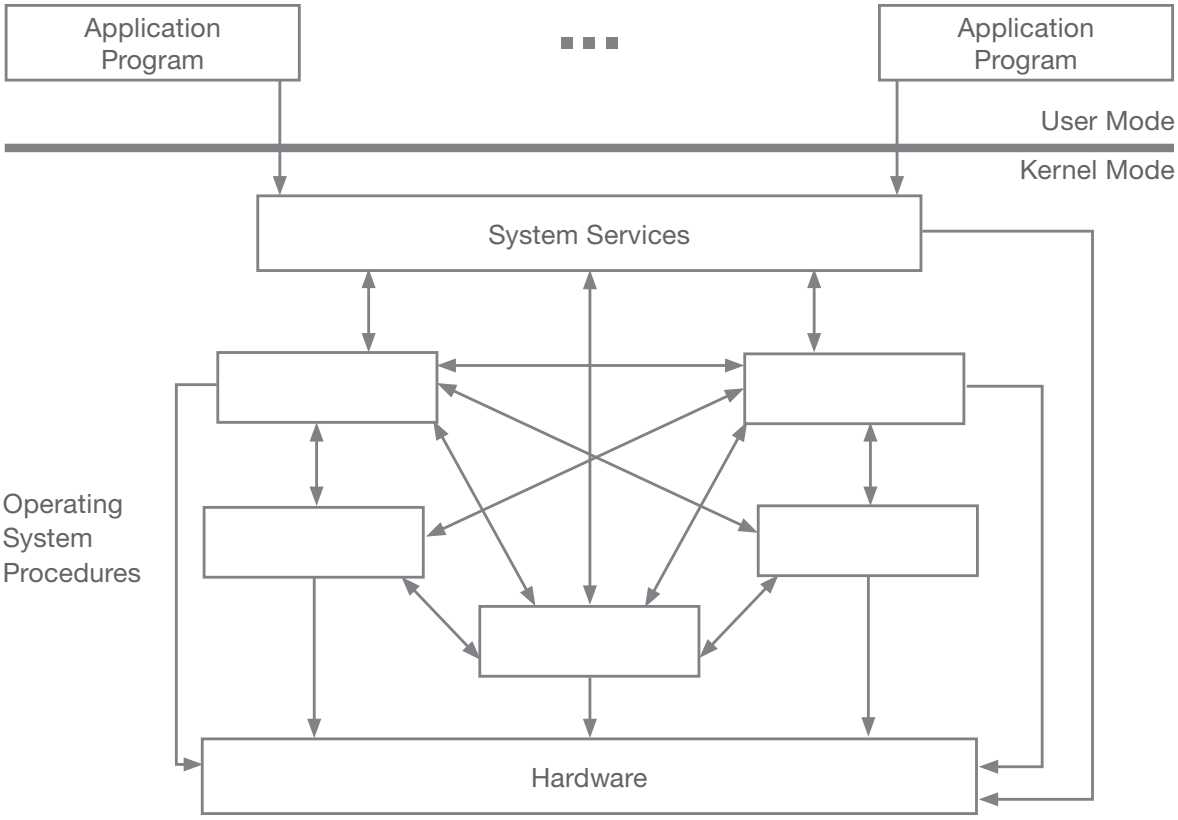

Abb. 2.1 Monolithisches System

In einem monolithischen System werden Applikationen strikt vom Betriebssystem getrennt. Das Betriebssystem läuft in einem privilegierten Modus – besser bekannt als „kernel mode“ – mit exklusivem Zugang zu Daten und zur Hardware. Jede weitere Applikation läuft im nichtprivilegierten Modus, bekannt als „user mode“. Im user mode werden nur bestimmte Schnittstellen (Interface) zum System und zum Datenbereich bereitgestellt. Ein Aufruf auf Seiten des user modes resultiert im Normalfall immer in einen Aufruf einer Funktion im kernel mode. Der Aufrufende Prozess blockiert normalerweise so lange, bis der Kernel die Anfrage abgearbeitet hat. Der Aufbau eine monolithischen Systems führt zwar zu einem schnellerem und kompakteren System, die Absturzsicherheit ist jedoch kaum gegeben. Fällt eine Komponente aus, so kann sie nicht neu geladen werden, das System stürzt ab.

2.1.2 Schichtmodell (Windows NT, W2K, XP, Vista)

Entgegen dem monolithischen System werden hier größere Prozesse in Modulen verpackt. Jedes Modul unterstütz wiederum ein Set von Funktionen, die von den anderen Modulen aufgerufen werden können. Allerdings kann ein höher gelegener Layer (Schicht) nur Funktionen eines tiefer gelegenen Layers aufrufen. Der Vorteil ist die leichtere Wartbarkeit (einzelne Layer können normalerweise ausgetauscht oder modifiziert werden, ohne Änderungen an den restlichen vollziehen zu müssen) und der einfachere Aufbau der Software. Es ist zudem möglich das komplette System für eine Fehlersuche zur Laufzeit zu debuggen, von der untersten zur obersten Schicht.

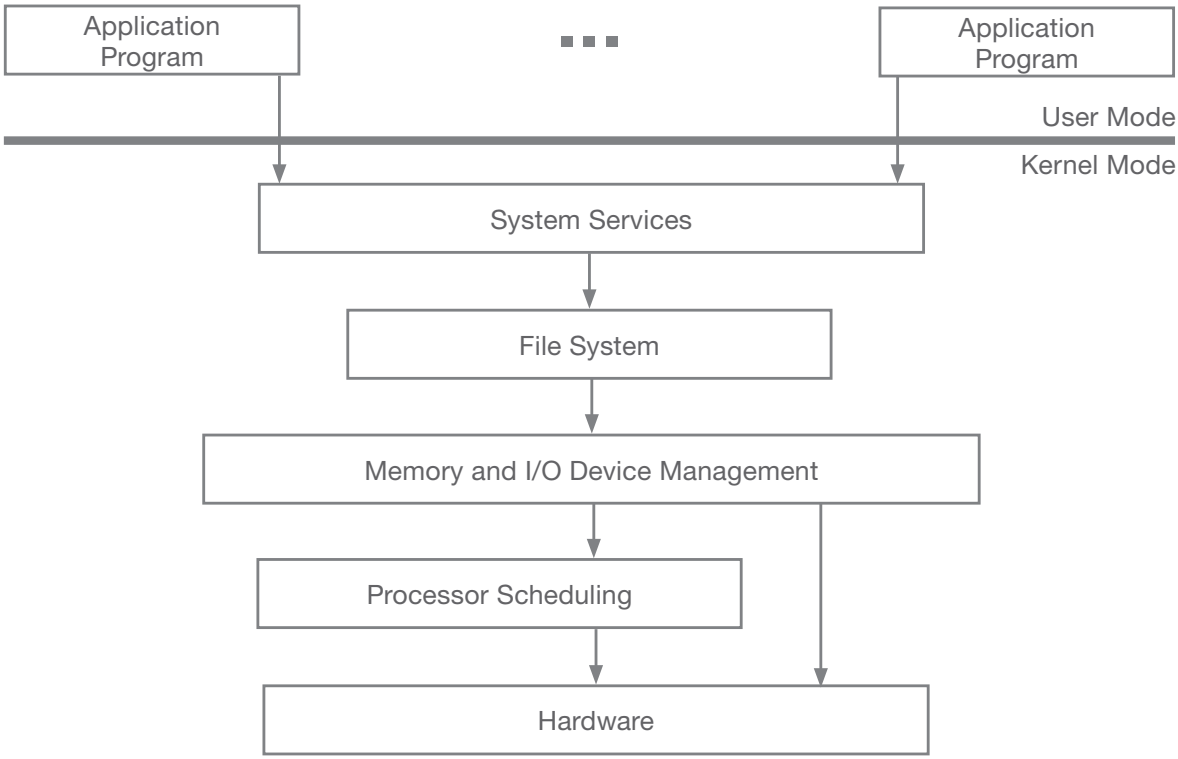

Abb. 2.2 Schichtmodell

GRUNDLAGEN

2.1.3 Client/Server (Windows NT, W2K, XP, Vista)

Dieses Modell hat wenig mit der Vernetzung von Systemen in einem Netzwerk zu tun, obwohl die Bezeichnung den Gedanken zulässt. Die Idee ist, das System in kleinere Prozesse aufzuspalten, welche jeweils eigene Funktionen bereitstellen (z.B.: Scheduling, Speicherverwaltung, etc.) Theoretisch könnten auch Hardware-devices als Prozess gesehen werden. Client und Server Prozesse laufen im user mode, wobei der Serverprozess in einer steten Schleife prüft, ob ein beliebiger Client eine Anforderung stellt. Der Client, welcher als weiterer Systemprozess, Applikation, etc. auftreten kann, sendet eine Nachricht an den Server. Diese wird jedoch nicht direkt versendet, sondern durch den Microkernel empfangen und weitergeleitet. Der Server führt die Anweisung aus und schickt das Ergebnis mittels Microkernel an den Client zurück.

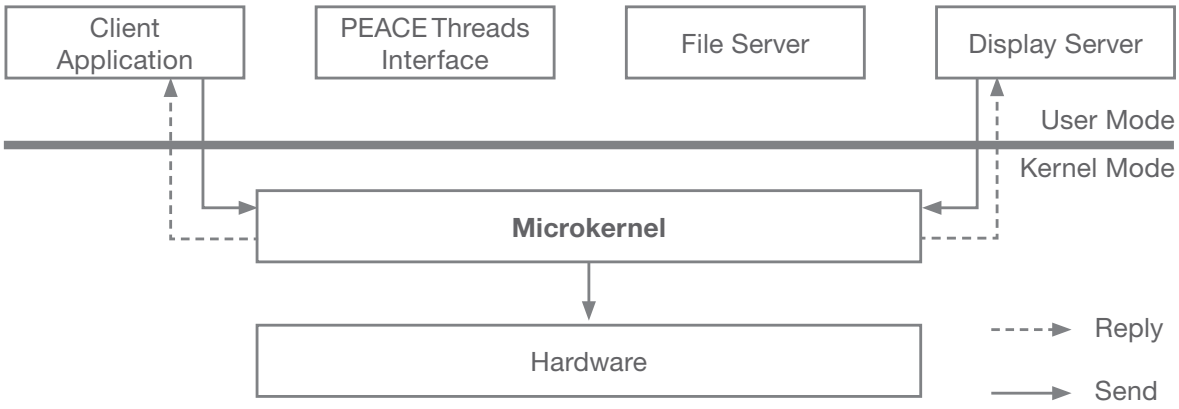

Abb. 2.3 Client/Servermodell

Der Vorteil dieses Aufbaus ist die Ausfallsicherheit. Es ist jederzeit möglich, dass ein Prozess, egal ob Server oder Client, ausfällt, er kann dann neu gestartet werden, ohne dass andere Prozesse dies bemerken. Das Betriebssystem ist dadurch vor Verklemmungen oder Deadlocks geschützt. Der Client muss zudem nicht wissen, ob und wo der gesuchte Server existiert, es ist somit möglich den Server in einem verteilten System laufen zu lassen, der Client merkt auch hiervon wieder nichts.

2.1.4 Windows NT Strukturmodell

Die Windows Familie – basierend auf NT –, nutzt die letzten zwei Modelle. Der kernel mode wird meist „NT executive“ genannt. Das Schichtenmodell kommt nun zum Zuge, der Kernel wird in drei Schichten aufgeteilt: Executive Services, Microkernel und Hardware Abstraction Layer (HAL).

Executive Services: Hier werden eine Reihe von Funktionen zusammengefasst, die für die Speicherverwaltung, Sicherheit, Ein/Ausgabe, etc. genutzt werden. Die einzelnen Prozesse interagieren untereinander nach dem o.g. Client/Server Modell.

Microkernel: Hier findet das Scheduling von Threads, die Behandlung von Exceptions und Interrupts, Synchronisation von Multiprozessorsystemen, etc. statt. Der Scheduler ist eine Instanz, welche die verfügbaren Ressourcen sinnvoll für die einzelnen Prozesse zur Verfügung stellt und wieder entzieht. Das Scheduling funktioniert zunächst nach Prioritäten, die von einem Entwickler vorgegeben werden. Die Prioritätstufen sind „Niedrig“(Ruhezustand), „Weniger als Normal“, „Normal“, „Höher als Normal“, „Hoch“ und „Echtzeit“, wobei die Einstellung „Echtzeit“ trotz des Namens keine harten Echtzeitanforderungen in einem Windowssystem ermöglicht und in der Regel auch möglichst wenigen Systemprozessen vorbehalten sein soll. Standardwert ist für alle Programme „Normal“, die Priorität kann jedoch zum Start oder während der Laufzeit einer Applikation über den „Windows Task Manager“ verändert werden. Die Applikation stellt hierbei den „Mutterprozess“ dar, von ihm gestartete weitere Prozesse/Threads erben automatisch die Priorität des Mutterprozesses. Die Priorität wird intern als ganzzahliger Integerwert dargestellt. Somit hat die Priorität „Normal“ den Basiswert „8“. Der Scheduler kann nach bestimmten internen Algorithmen den Prioritätslevel verändern, abhängig von dem Basiswert. Ein zweiter Faktor für das Scheduling ist der sogenannte „Quantum“ Wert. Dieser wird vom Scheduler dynamisch vergeben und bestimmt die Prozessorzeit, welche dem Thread zur Verfügung gestellt wird (Unabhängig davon, ob der Thread dabei seine Aufgaben abarbeiten kann). Auf diese Zeit hat ein Entwickler keinen Einfluss, sie stellt eine unberechenbare Größe dar und es ist somit nicht vorhersehbar, wann ein Thread die ihm beauftragten Berechnungen/Aufgaben fertig stellt [SCHED].

GRUNDLAGEN

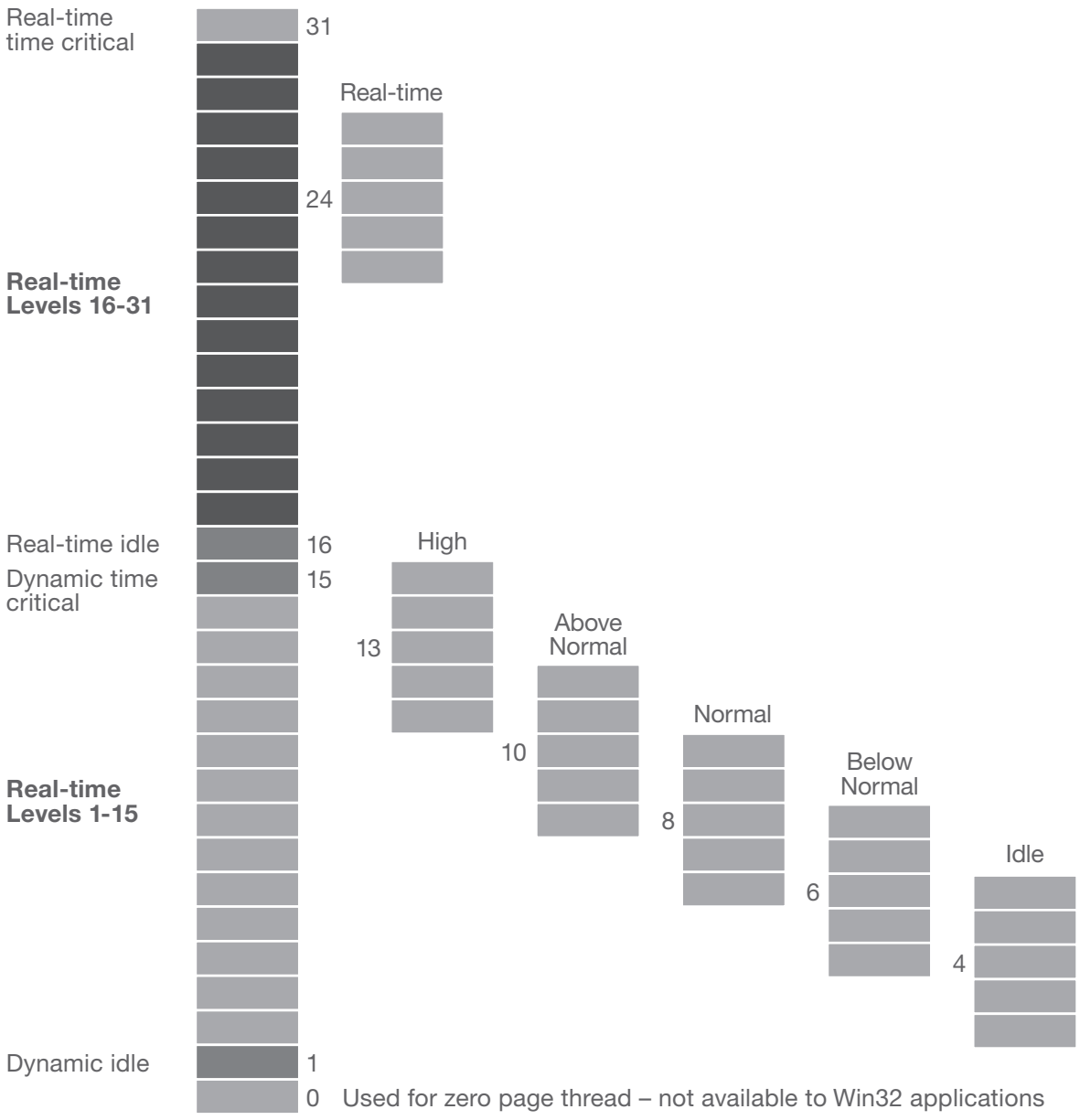

Abb. 2.4 Schema zu den Prioritäten unter Windows

Hardware Abstraction Layer (HAL): Dieser Layer trennt das Betriebssystem von der eigentlichen Hardware, agiert als Interface zur Hardware und schützt somit das System vor Komplikationen mit der Interaktion selbiger.

GRUNDLAGEN

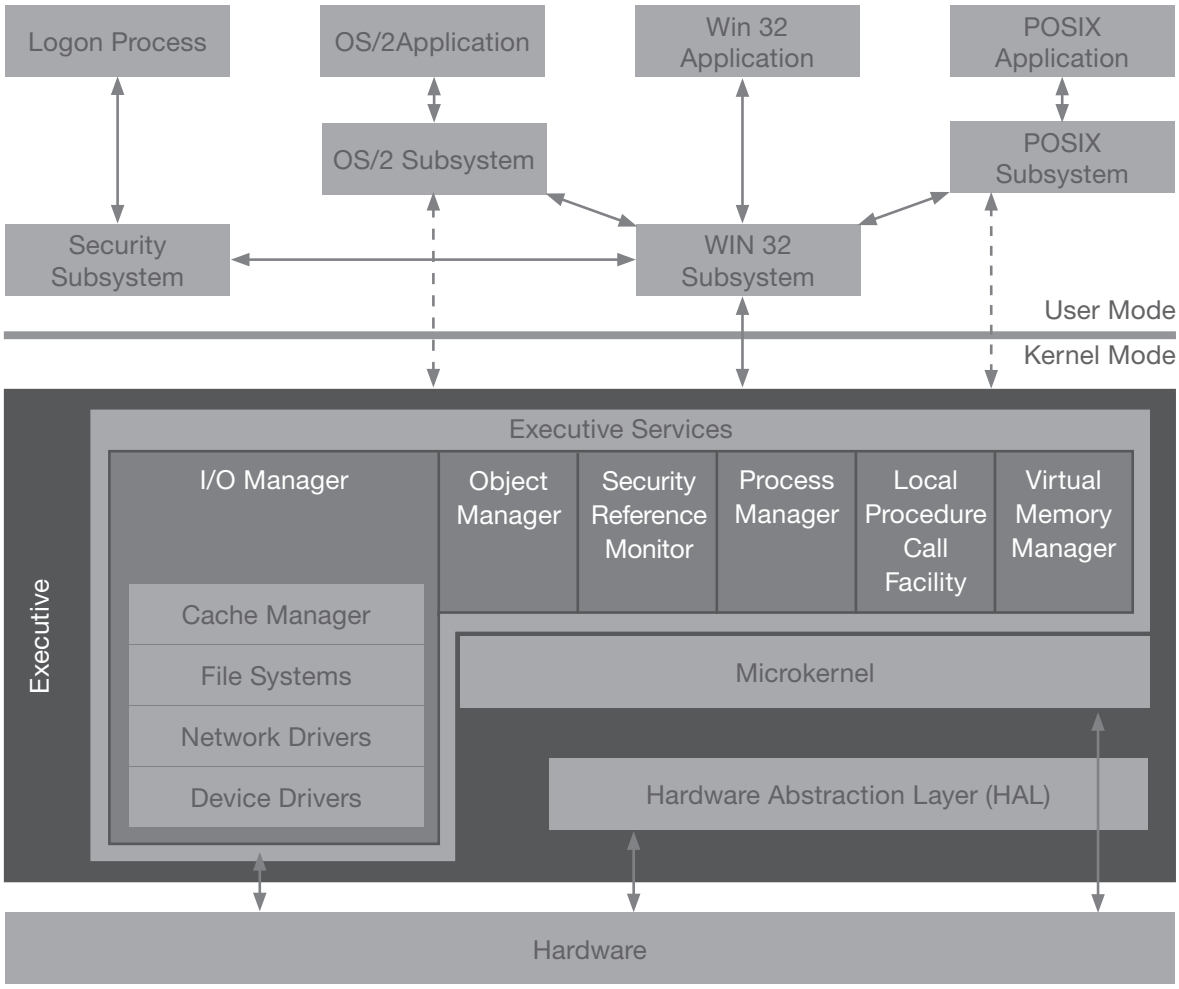

Abb 2.5 Windows NT Blockdiagramm

Nachdem das Betriebssystem nun in seinem Aufbau und seiner Funktionsweise etwas bekannt ist, lässt sich nun abschätzen, wo Verzögerungen in der Ausführung eines „einfachen“ Schreibprozesses auf die serielle Schnittstelle aufkommen werden. Das zu erstellende Mess-System muss sich, wie alle weiteren Applikationen auch, folgendem Ablauf unterwerfen:

Die Applikation ruft für eine Ausgabe an die serielle Schnittstelle die Methode comWrite() auf. Entweder erfolgt nun eine Weiterleitung der Anfrage an ein Subsystem oder an eine „dynamic-link library“ (.dll). In beiden Fällen führt dies zum Aufruf der Local Procedure Call Facility (LPC), welche für die Kommunikation aller lokaler Systemprozesse nach dem Client/Server Prinzip zuständig ist. Die

GRUNDLAGEN

Anforderung wird dann zunächst an den I/O Manager weitergeleitet, welcher zuerst eine Anfrage beim Object Manager – alle Systemkomponenten, Applikationen, etc. werden unter NT als Objekt dargestellt und müssen sich während der Initialisierungsphase beim Object Manager anmelden – durchführt. Der Object Manager prüft, ob das Zielobjekt (hier: serielle Schnittstelle) überhaupt vorhanden ist. Danach muss eine Anfrage beim Security Reference Monitor gestellt werden, ob das aufrufende Objekt (hier die DLL) überhaupt die nötigen Rechte besitzt um die anvisierte Aktion auszuführen. Ist soweit alles fehlerfrei erfolgt, erstellt der I/O Manager ein sogenanntes „I/O request packet (IRP)“. Dieses wird an den jeweiligen Gerätetreiber weitergeleitet und nach Erledigung der Aufgabe an den I/O Manager zurückgegeben. Dieser vernichtet dann das IRP und gibt an seinen Aufrufer einen Status zurück (success/failed).

Anhand der dargestellten Abläufe wird ersichtlich, dass der Weg zur seriellen Ausgabe nicht ohne Weiteres durchführbar ist. Bedingt durch die verschiedenen Aufrufe mehrerer Komponenten ergibt sich die windowseigene Latenzzeit. Dass diese nicht immer gleich ist – bedingt durch den Scheduler des Microkernels – wird im Verlauf dieser Arbeit noch ersichtlich. Letztendlich entscheidet der Scheduler anhand der dynamischen Prioritäten, wann ein Prozess zur Ausführung kommt oder pausieren muss.

2.2 Bluetooth – SPP Protokolle & AT-Kommandos [WIKI, BT1, BT2, BT3, BT4, BT5]

Der nach dem dänischen Wikingerkönig Harald Blåtand (zu deutsch Blauzahn, englisch Bluetooth) benannte Funkstandard wurde zu Beginn der 90er ursprünglich von Ericsson nach dem Standard IEEE 802.15.1 erstellt. Der Hauptzweck dieser drahtlosen Kommunikation bestand in dem Ersatz des Kabelgewirrs, welches sich derzeit schon erheblich vergrößert hatte. Seinerzeit mussten Tastaturen, Mäuse, Drucker etc. bereits per Kabel angebunden werden. Mit dem neu aufkommenden Trend Digitalkameras, PDAs, etc. mit dem PC zu verbinden wurde das Interesse an einer zuverlässigen und günstigen Funktechnologie groß. Die Funktechnologie ermöglicht es zudem drahtlose Ad-Hoc Pikonetze aufzubauen. Pikonetze sind lokale Netze, die in der Regel über keine große Ausdehnung verfügen und keine

GRUNDLAGEN

Infrastruktur benötigen. Jedes Gerät kann mit bis zu sieben weiteren Geräten innerhalb des Netzes kommunizieren, wobei es maximal einen Master geben darf. Daneben besteht auch die Möglichkeit, Geräte in mehreren Pikonetzen zu betreiben, hier allerdings nur Slaves. Der Master kann jedoch auch als Slave in einem weiteren Pikonetz agieren, dabei wird das alte Netz unterbrochen.

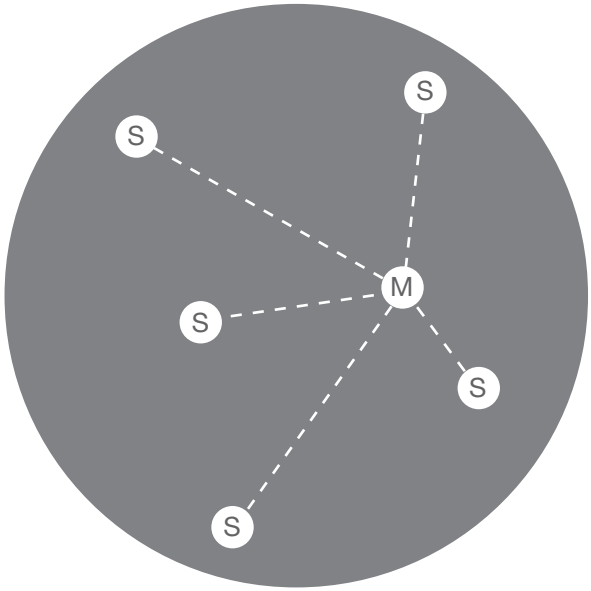

Abb. 2.6 Aufbau eines typischen Pikonetzes

Im Folgenden werden die Bluetoothspezifikationen und Grundfunktionen grob vorgestellt, weiterhin wird näher auf das für das Mess-System relevante Protokoll und die Steuerbefehle eingegangen.

2.2.1 Frequenzbereich 2,4 GHz, ISM-Band

Das Bluetooth-System wird im lizenzfreien 2,4-GHz-ISM-Band (Industrial, Scientific and Medical-Band) betrieben (2.400 bis 2.483,5 MHz). Das Frequenzband wird in 79 Hochfrequenzkanäle aufgeteilt, welche von 0 bis 78 durchnummeriert sind. Sie liegen in Abständen von 1 MHz und beginnen bei 2.402 MHz. Am Anfang und Ende des Bandes sind Pufferzonen eingerichtet, um länderspezifische Vorschriften einhalten zu können. Daher geht das real nutzbare Band von 2.402 bis 2.480 MHz. Die Datenrate liegt bei ca. 1 Mbit/s, wobei die Nettodatenrate je nach

GRUNDLAGEN

Paketgröße und dadurch variablem Overhead von Start/Stop-bits etc. geringer ist. Bluetoothmodule werden in zunächst drei Leistungsklassen aufgeteilt:

Geräteklasse	Ausgangsleistung Max	Ausgangsleistung Min.	Reichweite
Class 1	100 mW	1,00mW	~100m
Class 2	2,5 mW	0,25mW	~ 50m
Class 3	1 mW	-	~ 10m

Abb 2.7 Bluetoothgeräteklassen
Die Reichweiten hängen jedoch von der Antennenart und der Umgebung ab (Mauern, etc.).

Bluetooth bietet zudem eine Option zur Verschlüsselung (128bit) der Daten an, der Anwender kann sich hier eines eigens gewählten Schlüssels (PIN) bedienen, der mindestens 4, höchstens 16 Ziffern oder Zeichen nach UNICODE 8 enthalten darf. Dieser PIN muss neben dem Master auch den Slaves bekannt sein. Der fertige Sitzungsschlüssel für die sichere Übertragung wird schließlich aus dem gewählten PIN-Code, einer vom Master erstellten Zufallszahl und der Geräteadresse erzeugt, wobei die Zufallszahl den Slaves für die Entschlüsselung mitgeteilt werden muss. Bei jeder neu auftretenden Übertragung wird dieser neu erzeugt.

2.2.2 Frequenzsprungverfahren (frequencyhopping)

Die Frequenzkanäle werden von den Geräten nach einem bestimmten Algorithmus „durchwandert“ (Frequenz-Hopping-Algorithmus oder auch „Hop-Frequenz“). Dieser Algorithmus ist für jedes Modul vordefiniert, man unterscheidet aber hierbei zwischen dem Algorithmus zum Verbindungsaufbau und dem für den regulären Betrieb. Während des Verbindungsaufbaus wird ein statischer Algorithmus durchlaufen, hier werden lediglich 32 der 78 möglichen Kanäle genutzt um Zeit zu sparen. Als erstes werden die 16 benachbarten Kanäle verwendet:

$\{f(k - 8), \ldots, f(k), \ldots, f(k + 7)\}$,

wobei f(k)= der zum Startzeitpunkt der Hopsequenz aktuelle Kanal ist.

Danach die restlichen Kanäle: $\{f(k + 8), \ldots, f(k + 15), f(k - 16), \ldots, f(k - 9)\}$

GRUNDLAGEN

Die Menge der Kanäle wird in zwei sogenannte „Trains“ unterteilt, die erste Menge stellt den A-train dar, die Zweite den B-train. Die Abtastung der Trains wird ständig wiederholt, wobei in der Regel der Train alle 2,25 Sekunden gewechselt wird. Die Dauer der gesamten Abtastprozedur ist nicht limitiert, wird meist aber auf 5 Sekunden begrenzt. Wozu aber teilt man die Frequenzen in Trains auf? Dies wird anhand eines realistischen Szenarios erläutert. Es existieren zwei Module, A und B, welche bereits verbunden und damit synchronisiert sind. Die Verbindung wird getrennt, beide Module durchlaufen nun wieder die 32 oben beschriebenen Frequenzen. Wenn nun ein Modul sich erneut mit dem anderen Modul verbinden will, so ist die Chance nahezu bei 100%, dass es das Gerät auf den benachbarten 16 Kanälen findet. Durch die nie genau taktgleichen Quarze, welche die Frequenz der Hardware vorgeben, driften nach einiger Zeit die ehemals synchronisierten Module in der Sprungfrequenz auseinander. Werden die Module nicht vom Strom getrennt oder resettet, dauert es ca. 2,5 Tage, bis die Abdrift so hoch ist, dass sich das Zielgerät in den äußeren Frequenzkanälen befindet.

Der Verbindungsaufbau ist für alle Geräte gleich und kann mit dieser Methode ohnehin schon bis zu 10 Sekunden und länger dauern. Hat der Master einen kommunikationsbereiten Slave gefunden, gibt der Master nun seine aktuellen Timing-Informationen für den Verbindungsaufbau dem Slave bekannt. In diesen Informationen sind die vollständige Geräteadresse, die Phase der Hopping-Sequenz, die Systemzeit, eine Zufallszahl und die Clock-Informationen enthalten. Mit Hilfe der Clock-Information und der Systemzeit synchronisiert sich der Slave auf die selbe Hop-Frequenz des Masters. Mit Hilfe der Geräteadresse, der durch den Master generierten Zufallszahl und der Systemzeit kann der Slave nun den neuen Hop-Algorithmus für die bestehende Verbindung errechnen. Der grundsätzliche Berechnungsalgorithmus ist immer gleich, das Ergebnis jedoch durch die oben angegebenen Parameter unterschiedlich. Steht die Verbindung, sendet der Master ein zur Synchronisation bestimmtes Paket an den Slave, welches dieser quittiert. Erfolgt die Quittierung, kann der Master davon ausgehen, dass der Slave die Tabelle mit den Hop-Frequenzen richtig errechnet hat und nun synchron mit dem Master durch die Kanäle springt. Die Kanäle werden alle 625 Mikrosekunden(µs) gewechselt, das ergibt 1600 Hops pro Sekunde.

GRUNDLAGEN

Nach dem TDD-Verfahren (Time Division Duplex) wird die Zuteilung der Kanäle festgelegt, das heißt, dass Sender und Empfänger abwechselnd eine Sendeberechtigung haben. Der Master nutzt hier immer die geraden, der Slave immer die ungeraden Slots (Zeitfenster). Nach jedem Datenpaket, das gesendet wird, wartet das Gerät auf eine Antwort. Dadurch kann die Bandbreite am effizientesten genutzt werden, es ist so möglich die Paketgröße zu variieren und somit dem Up- und Downstream verschiedene Bandbreiten zuzuteilen.

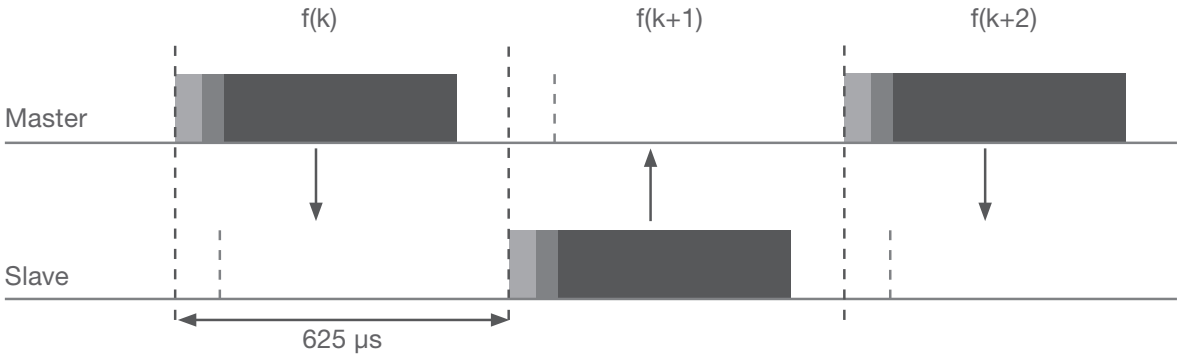

Abb 2.8 Slotbelegung nach TDD

Durch die Frequenzsprünge ist es möglich eine sichere Datenübertragung zu schaffen (der Hop-Algorithmus ist nur dem Master und den verifizierten Slaves bekannt) und man umgeht das Problem der Störanfälligkeit durch andere funkende Geräte in der näheren Umgebung im gleichen Frequenzband. Seit der Bluetoothspezifikation 2.0 wurde dieses Verfahren noch verbessert, das adaptive Frequenzhoppingverfahren blendet nun bei Übertragungsfehlern die gestörten und wohl anderweitig belegten Kanäle zur weiteren Nutzung aus. Der Master ermittelt dies anhand der Anzahl fehlerhafter Pakete und schlechter Signalstärke auf der jeweiligen Frequenz. Er blendet nun diese Frequenz aus seiner Sprungtabelle aus und teilt dies auch dem Slave mit. Die Liste der nutzbaren Frequenzen kann jedoch nur bis auf eine Mindestgröße von 20 Frequenzen runtergekürzt werden, da sonst das Frequenzhoppingverfahren sinnlos wird. Es wurde jedoch nicht in der Bluetoothspezifikation festgelegt, wann ein zunächst ausgeblendeter Kanal wieder freigegeben wird. Dies kann entweder anwenderspezifisch oder vom Chiphersteller definiert werden, hier ist z. B.: ein Algorithmus zu implementieren, der stets die Qualität der Kanäle testet und protokolliert.

GRUNDLAGEN

In jedem Zeitfenster (Slot) können Daten gesendet werden, wobei der Link Manager je nach Verbindungsqualität entscheidet, welche Paketgröße aktuell gewählt wird. Es gibt Paketgrößen von 1-Slot bis zu 5-Slot und theoretisch mehr. Da der Sender bei einem Mehrslotpaket den Overhead (Start-/Stoppbit, Checksumme, etc.) weglässt um die volle Zeit zum maximalen Transfer auszunutzen, kann es jedoch bei Datenverlusten zu erheblichen Einbußen bei der Datenrate kommen. Der Empfänger weiß erst mit dem Empfang des letzten Pakets, ob alle vorangegangenen Pakete fehlerfrei waren, oder ob es einen Übertragungsfehler gab. Leider können nicht einzelne Paketfragmente erneut gesendet werden, sondern nur das komplette Paket. Problematisch bei Mehrslotpaketen ist auch, dass die Frequenz während der Übertragung nicht gewechselt wird. Störungen des Kanals werden – wie oben beschrieben – erst nach Senden des Paketes bemerkt.

2.2.3 Der Bluetoothstack

Aus dem genannten Bluetoothstack werden nur die für diese Arbeit wichtigen Teile nachfolgend ausführlicher dargestellt. Der Stack ist das grundlegende Betriebssystem für ein Bluetoothmodul.

- **Bluetooth Radio**
  Spezifiziert das Frequenzband, die Sender- und Empfängereigenschaften (z. B. Sendeleistung) sowie die grundlegenden Eigenschaften der Funktechnik.

- **Baseband**
  Fast alle wichtigen Technologien, wie der Aufbau von Pikonetzen, Frequenzhopping, Adressierung von weiteren Bluetoothgeräten, Systemsicherheit, Fehlerkorrektur, etc., werden im Basisband implementiert.

- **Link Manager (LM)**
  Der LM sorgt für einen reibungslosen Verbindungsaufbau und ist verantwortlich für die Datensicherheit, Paketgröße und die Identifikation. Er ist in der Regel ein Teil der Firmware, welche mit dem Chip erworben wird.

GRUNDLAGEN

- **Host Controller Interface (HCI)**
  Das HCI bildet die Schnittstelle zwischen den unteren Protokollschichten (Bluetooth Controller) und den oberen Schichten (Host System), sofern der Bluetoothstack nicht exklusiv auf dem Bluetoothchip implementiert ist. Die spezifizierten Bluetoothprotokolle gehen damit durch die gesamte Hardware, wobei die unteren Layer auf dem Chip selbst und die oberen normalerweise auf einem anderen System (Coprozessor) liegen. Damit diese getrennten Systeme miteinander kommunizieren können wird das HCI eingesetzt. Der Sinn dabei ist, dass so die oberen Protokollschichten mit Chips verschiedener Hersteller genutzt werden können oder, dass z.B. der Host Controller in den Sleep Modus gehen kann um so Strom zu sparen.

- **Logical Link Control and Adaption Protocol (L2CAP)**
  Protokoll der Sicherungsschicht im Bluetoothschichtenmodell. Passt die höheren Schichten an die Fähigkeiten des Basisbands an und verbirgt die Übertragungsdetails (verbindungslos und verbindungsorientiert). Es stellt die Schnittstelle zwischen höher liegenden Anwendungsschichten und den tiefer liegenden Schichten der Firmware dar. Ohne diese Schicht ist zwar der grundsätzliche Bluetoothbetrieb möglich, jedoch ohne die wirklich netten Features wie Multi-Protokoll-Übertragung, Ad-Hoc-Netzwerke, etc. In der Abbildung 2.3 wird die Lage dieser Schicht ersichtlich.

- **Service Discovery Protocoll (SDP)**
  Das SDP legt die Sichtbarkeit eines Gerätes fest und welche Dienste es nach außen anbieten kann, quasi die „gelben Seiten“ des Bluetooth. Ein näheres Beispiel für solch eine Eintragung findet sich in einem Folgekapitel.

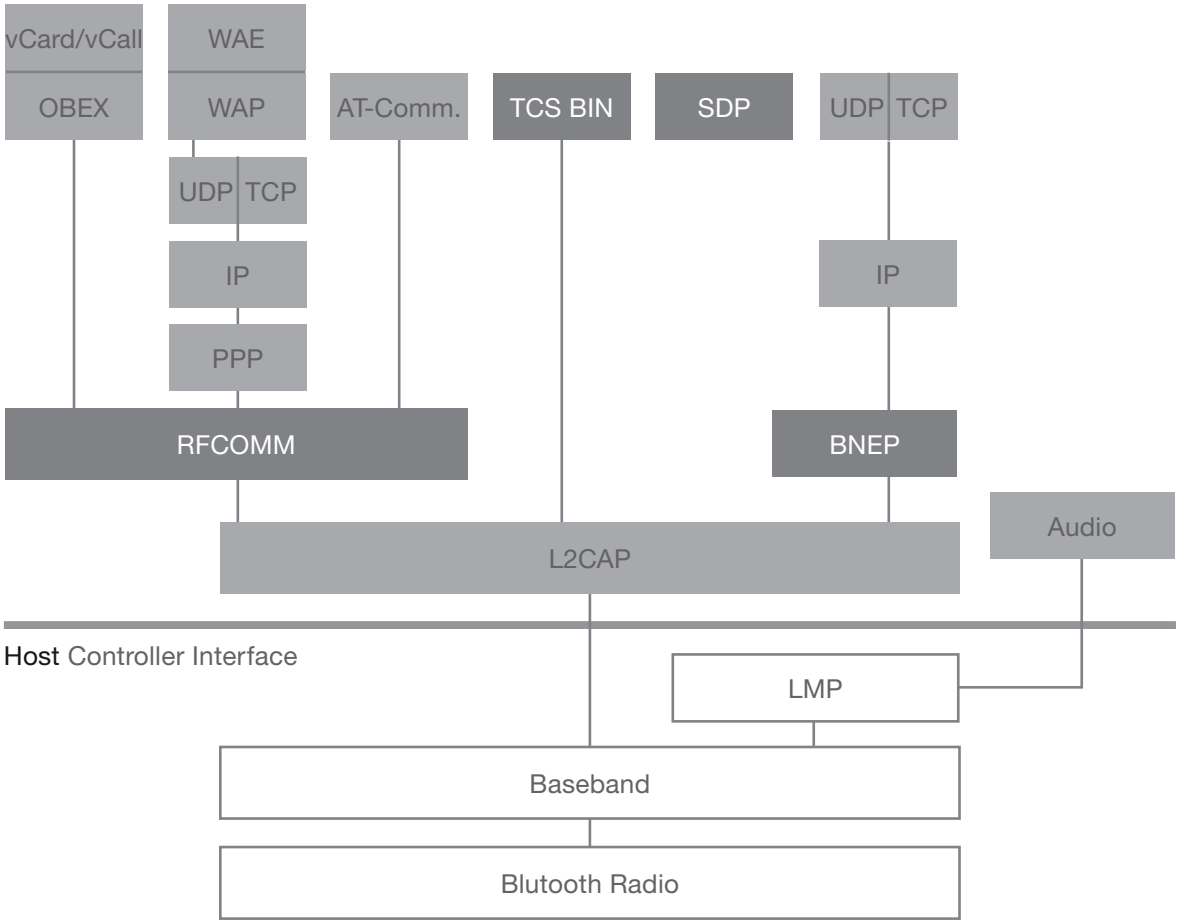

Abb 2.9 Bluetooth Protocol Stack

2.2.4 Das RFCOMM Cable Replacement Protocol

Das RFCOMM Kabelersatzprotokoll liegt auf dem Hostsystem und stellt dem Betriebssystem Dienstleistungen zur Verfügung. Es spielt eine zentrale Rolle bei der Verwendung von Bluetooth, da es die sonst übliche physikalische serielle Schnittstelle abbildet. Es können durch das RFCOMM bis zu 60 virtuelle Schnittstellen nach dem RS232-Standard mit einer Datenrate von ca. 128kbs simuliert werden. Diese Geschwindigkeit entspricht damit theoretisch zwei gebündelten ISDN-Kanälen. Es bietet nach der Norm EIA/TIA-232-E zusätzlich Hardwarehandshake und eine maximale Geschwindigkeit von 115kbps. Das Protokoll basiert auf dem ETSI Mobilfunkstandard TS 07.10 um die Portierbarkeit auf

GRUNDLAGEN

Handys jederzeit zu ermöglichen. Lediglich eine kleine Untermenge ist jedoch implementiert.

Es gibt zwei Arten für die Kommunikation über die virtuelle serielle Schnittstelle:

- **Typ1 Kommunikation:** Punkt zu Punkt Verbindung zweier gleichberechtigter Geräte. Typ 1 Geräte sind Endgeräte, also Mäuse, Drucker, Computer, etc.
- **Typ 2 Kommunikation:** Gateway-Verbindung, hierdurch wird es PCs ohne eigene Bluetoothmodule ermöglicht, auf entfernte Typ 1 Geräte zuzugreifen. Typische Vertreter dieses Typs sind Accesspoints oder Modems, die beispielsweise ein Netzwerk von Geräten mit entfernten Endgeräten via Bluetooth verbindet. In der folgenden Grafik ist beispielsweise ein PC über den Gateway mit einem entfernten Endgerät (Drucker, etc.) verbunden, der nur über Bluetooth zugänglich ist.

Abb. 2.10 Beispiel für eine Typ 2 Kommunikation

Gemäß der TS 07.10 wird für die Kommunikation zwischen zwei Geräten eine Null-Modem-Emulation erzeugt. Im Prinzip sollte das Null-Modem-Modell immer funktionieren, es kann aber nicht garantiert werden – wie auch bei kabelgebundenen Verbindungen. Dies ist somit keine Schwäche des Bluetooth, sondern ein generelles Problem einer Null-Modem Verbindung.

GRUNDLAGEN

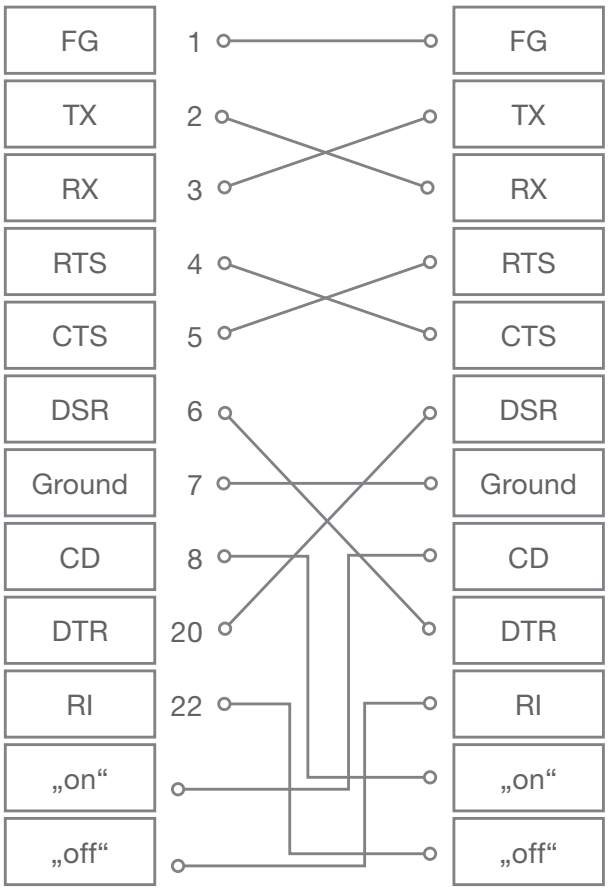

Abb 2.11 RFCOMM-Null-Modem-Schema

Geräte, die über das RFCOMM verbunden sind, können theoretisch bis zu 60 serielle Verbindungen gleichzeitig öffnen. Dies ist jedoch implementierungsabhängig und daher nicht unbedingt realisierbar (z. B.: muss die Software in der Lage sein mehrere RFCOMM Instanzen auf dem Gerät zu erzeugen). Für die Identifikation der einzelnen gemultiplexten Kanäle wird schließlich ein Data Link Identifier (DLCI) verwendet. Der DLCI ist Teil des Adressfeldes innerhalb eines TS 07.10 Rahmens und bleibt während der Verbindungssession immer gleich.

RFCOMM definiert also ein Protokoll, das – auf Basis der TS 07.10 Spezifikation – die Emulation einer seriellen Schnittstelle ermöglicht. Wird dieses Protokoll von einem Bluetoothstack unterstützt, so bedeutet es im Umkehrschluss nicht, dass auch alle Anwendungen, die eine serielle Schnittstelle nutzen, problemlos mit Bluetooth zusammenarbeiten werden. RFCOMM wird meistens ein Teil eines

GRUNDLAGEN

Schnittstellentreibers sein, welcher die eigentliche Port-Abstraktion vornimmt. Jede Applikation muss sich zudem über das SDP registrieren bevor es die serielle Schnittstelle verwenden kann.

2.2.5 Das SPP Profil (Serial Port Profile)

In der Bluetoothspezifikation sind 13 verschiedene Profile definiert, teilweise aufeinander aufbauend, welche alle im RFCOMM enthalten sind. Die vier grundlegenden Profile sind

- **GAP (General Access Profile)**
  Hier werden allgemeine Prozeduren für die Erkennung von Geräten, den Verbindungsaufbau und das Verbindungsmanagement bereitgestellt. Es stellt das Grundlegende Profil für die Folgenden dar.

- **SDAP (Service Discovery Application Profile)**
  Dieses Profil beschreibt grundsätzliche Funktionen für Anwendungen, die auf das SDP zugreifen um verfügbare Profile/Protokolle zu proklamieren.

- **SPP (Serial Port Profile)**
  Profil, welches zum Einsatz kommt, sobald Bluetooth als Kabelersatz dienen soll. Auf SPP basieren z.B.: die Profile für Fax- oder LAN-Dienste.

- **GOEP (Generic Object Exchange Profile)**
  Für den Austausch komplexer Datenobjekte kommt das GOEP zum Einsatz. Es definiert die Verwendung von OBEX innerhalb von Bluetooth und im Besonderen den Dateitransfer und die Synchronisation von Datenobjekten.

GRUNDLAGEN

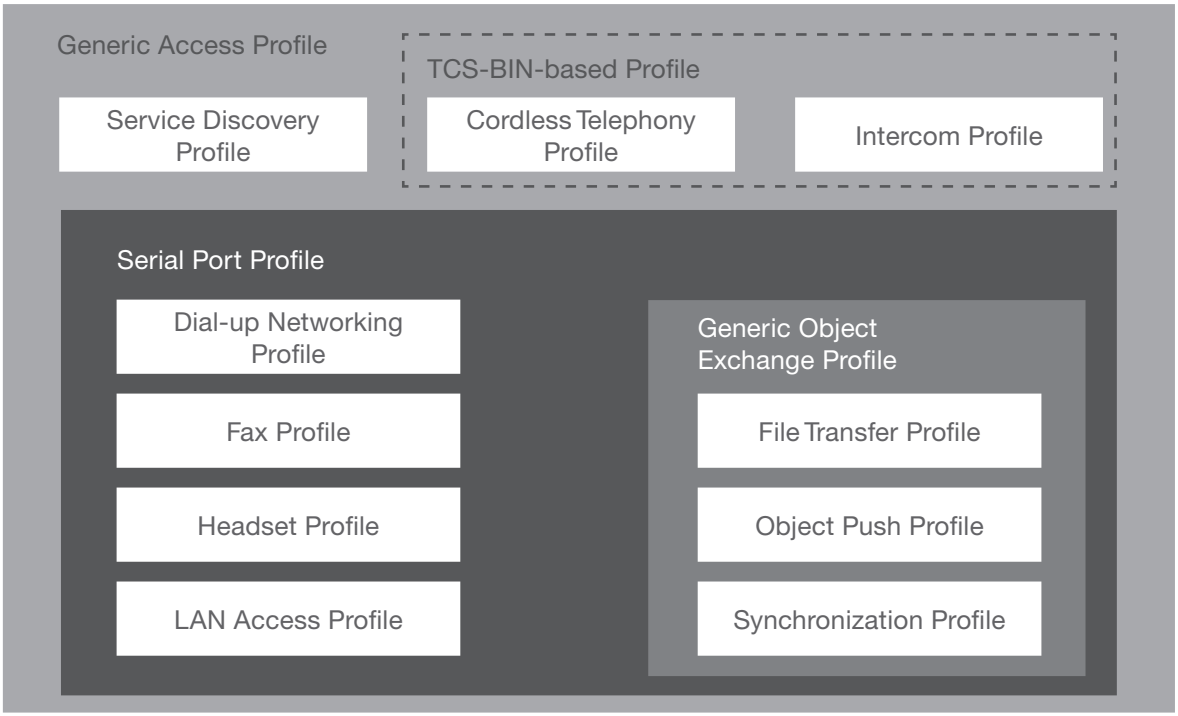

Abb 2.12 Aufbau des GAP

Für diese Arbeit wird lediglich das SPP Profil bearbeitet, der normale Umfang der kompletten Profilbeschreibung beträgt alleine in der Kurzfassung aus der Bluetoothspezifikation 1.1 für alle 13 Profile rund 450 Seiten. Sinnvoll zu erwähnen ist jedoch, dass das GAP im Modell ganz oben in der Profilhierarchie steht und alle weiteren Profile mehr oder weniger direkt auf diesem aufbauen.

Wenn serielle Datenverbindungen ersetzt werden sollen, kommt das Serial Port Profile zum Einsatz. Damit wird die Basis für die meisten Bluetoothanwendungen geschaffen. Alle weiteren Profile verwenden das SPP, außer das Telefon-Steuerungs-Profil. Typische Anwendungen, welche die serielle Verbindung nutzen werden, sind sogenannte „Legacy Application“, also Anwendungen, die bisher über einen konventionellen seriellen Kanal kommuniziert haben und nun auf der Basis von RFCOMM einen drahtlosen Datenaustausch vollziehen können. Ein Problem wird hier bereits erkennbar, welches auch in dieser Arbeit thematisiert werden soll: Die Tatsache, dass eine Funkverbindung immer erst aufgebaut werden muss vor einem Datenaustausch steht dem bisherigen Fall gegenüber, dass ein physikalisches Kabel mit dem Zusammenstecken die Verbindung bereits aufgebaut hat.

GRUNDLAGEN

Es entstehen dadurch nicht nur Verzögerungen, es muss nun auch sichergestellt werden, dass die Verbindung zunächst einmal aufgebaut wird. Die hierfür nötige Software ist je nach Anwendungsfall und Hardware durchaus unterschiedlich, der grundsätzliche Aufbau mit den nötigen Funktionen ist jedoch immer gleich.

Bei der seriellen Verbindung zweier Bluetoothgeräte gibt es immer einen Master, oder auch Initiator. Dieser nimmt die aktive Rolle in dem Aufbau der Kommunikation ein. Das zweite Gerät, der Slave oder Acceptor nimmt dagegen die passive Rolle ein. (Diese Unterteilung der Verhaltensweisen wird im späteren Verlauf der Arbeit noch eine kleine, jedoch nicht unwesentliche Rolle spielen.) Sicherheitsmerkmale, wie Autorisierung oder Verschlüsselung sind im Serial Port Profile optional, müssen jedoch bei einer Anforderung entsprechend verarbeitet werden.

Für das Serial Port Profile wird nur die Unterstützung von 1-Slot-Datenpaketen vorausgesetzt, wodurch die maximale Datenrate von 128 kbps zustande kommt. Dies ist nicht weiter tragisch, dient es doch der Sicherheit, dass auch bei schlechtem Empfang möglichst fehlerfreie Transfers möglich sind. Höhere Datenraten sind optional jedoch möglich bei Mehrslotdatenpaketen.

Der SPP-Dienst definiert schließlich drei verschiedene Dienstprozeduren: der durch den Initiator getriebene aktive Verbindungsaufbau und das Einrichten der virtuellen seriellen Verbindung, die Handlungen des Akzeptors zum Annehmen eines Verbindungsaufbaus und Einrichten der seriellen Verbindung und schließlich in welcher Form der Akzeptor seine Dienste in der SDP-Datenbank hinterlegt.

2.2.5.1 Aktiver Verbindungsaufbau

Der Verbindungsaufbau wird durch eine Anfrage an den SDP nach einem bestimmten Dienst initiiert. Mittels der dort gespeicherten Attribute wird festgelegt, über welchen RFCOMM Kanal die Verbindung aufgebaut wird. Soweit hier mehrere Möglichkeiten gegeben sind, kann der Anwender entscheiden, welche Schnittstelle zu nutzen ist. Optional kann auch eine gesicherte Verbindung angefordert werden. Schließlich kann ein neuer L2CAP Kanal für die RFCOMM Kommunikation und mit der ausgewählten Kanalnummer der Datenlink für den Datentransfer geöffnet werden.

2.2.5.2 Akzeptieren einer Verbindungsanfrage

Für die Anforderung einer gesicherten Datenverbindung muss der Akzeptor in der Lage sein die Verschlüsselung zu aktivieren. Des Weiteren muss er in der Lage sein einen L2CAP Kanal einzurichten und eine RFCOMM Sitzung zu unterstützen.

2.2.5.3 Registrieren der verfügbaren Dienste

Alle Applikationen, die über RFCOMM erreichbar sind, müssen in der SDP Datenbank registriert werden, damit andere Devices Kenntnis über die für den Verbindungsaufbau benötigten Protokolle erhalten. Anbei ein Beispiel eines Eintrags im SDP für das Serial Port Profile:

Gegenstand	Beschreibung	Type	Wert	AttributID
ServiceClassIDList ServiceClass()	Serial Port	UUID	0x1101	0x0001
ProtocolDescriptorList Protocol0 Protocol1	L2CAP RFCOMM	UUID UUID	0x0100 0x0003	0x004
ProtocolSpecificParameters	Serverkanal	Uint8	#Kanalnr.	
ServiceName	anzeigbarer Text	String	„COM5“	Ggf. Sprach-spezifischer Offset

Abb 2.13 Eintrag im SDP für das SPP-Protokoll

Die in der Spalte „Wert“ angegebenen Parameter sind für das SPP immer gleich (bis auf den ServiceName und die Kanalnummer, Letztere wird dynamisch zum Start vergeben, bleibt aber dann statisch). Sie dienen zur Identifizierung der benötigten Protokolle.

2.2.6 Die Bluetoothhardware [AT]

Für diese Arbeit wurden zwei Hardwaredevices der Firma Stollmann E+V GmbH eingesetzt. Es handelt sich um zwei Bluetooth Evaluation-Kits BlueEva+C11/G2.

Abb 2.14 BlueEva+C11/G2

Diese sind bereits für das SPP vorkonfiguriert und lassen sich über die RS232-Schnittstelle ansprechen. Das montierte Bluetoothmodul ist ein sogenanntes BlueMod+C11/G2, es handelt sich hierbei um ein Class 1 Modul nach der Bluetoothspezifikation 2.0 von CSR (Cambridge Silicon Radio) mit einem Atmel AT91SAM7. Wie bereits im Kapitel 2.2.3 angesprochen, ist hier der Bluetoothstack aufgespaltet. Auf dem Bluetoothchip sind die unteren Protokollschichten implementiert, während die oberen Schichten auf dem Atmelprozessor laufen. Verbunden werden beide Module über das HCI. Für einen praxisbezogenen Einsatz wäre der Atmelprozessor z.B. in einem Handy, PDA, etc. implementiert.

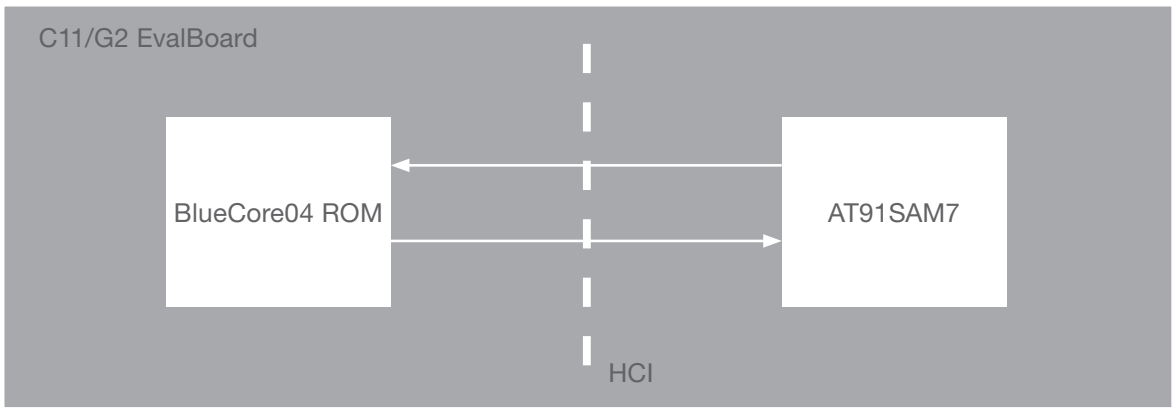

Abb. 2.15 Blockschaltbild des C11/G2 EvalBoard

GRUNDLAGEN

Angesteuert wird das Device über sogenannte AT-Befehle. Der AT-Befehlssatz wurde ursprünglich von der Firma Hayes Communications für die einfache Konfiguration von Modems entwickelt und ist zu einem Quasi-Industriestandard geworden. AT bedeutet hier „attention“, gefolgt von einem Befehl. Der gesamte Befehlssatz teilt sich in vier grundlegende Befehlsklassen auf:

1. Grundlegender Befehlssatz:
Nach dem AT folgen hier Buchstaben – manchmal auch zusätzlich in Verbindung mit Zahlen – z. B.: ‚D’ für „dial“. Oft folgen hier noch Nummern. Der vollständige Befehl für den Verbindungsaufbau lautet somit ATD{Btadress}. Die Bluetoothadresse ist die des Slaves, angegeben als zwölfstelliger Hexadezimalcode. Generell ist zu beachten, dass die Groß-/Kleinschreibung nicht relevant ist. Im ASCII Zeichensatz unterscheiden sich große und kleine Buchstaben nur durch ein Bit im Bitmuster, welches bei der Zeichenerkennung nicht beachtet wird.

2. Erweiterter Befehlssatz:
Durch die Zugabe des Sonderzeichen ‚&’ nach dem AT kann der Grundlegende Befehlssatz erweitert werden. Ein Beispiel ist hier AT&F für das Setzen der Fabrikeinstellungen, wobei ATF das Übertragungsformat festlegt.

3. Herstellerabhängiger Befehlssatz:
Diese Befehle beginnen mit ‚\’ oder ‚%’, hierunter verbergen sich proprietäre Befehle der Hersteller.

4. Registerbefehle
Hiermit können die Register selbst beschrieben werden. Die Syntax hat das Muster S(r)=n, wobei ‚r’ die Nummer des Registers und ‚n’ der neue Wert darstellt. Der Befehl S7=90 verändert den Wert des Registers #7 auf den Wert 90.

Ein ausführlicherer Auszug aus einer öffentlichen Hayes Command Liste befindet sich im Anhang.

GRUNDLAGEN

2.3 Die RS232-Schnittstelle [WIKI, RS232]

2.3.1 Hardware

Eine im Computerhardwarebereich stark verbreitete und bekannte Computerschnittstelle stellt die in den 1960er Jahren erfundene serielle Schnittstelle RS232 dar. Bis Anfang 2000 war sie immer noch standardmäßig auf allen erwerblichen Mainboards verbaut und ist heute in industriellen Steuerungsanlagen weiterhin vorzufinden. Die Schnittstelle wurde später umbenannt von der EIA-Electronic Industries Alliance zu EIA-232. Die EIA gibt bis heute weltweite Standards unter Anderem für Modemverbindungen frei. Obwohl schon vor einigen Jahren umbenannt, ist der Name „RS232“ unter Anwendern und Entwicklern am meisten verbreitet. Die Umbenennung ist eine Folge mehrerer Erweiterungen und Veränderungen des Standards, Zusammenschluss der Organisation mit anderen, etc. Somit kann die hier beschriebene Schnittstelle auch RS-232, EIA-232, TIA-232, EIA/TIA-232, ANSI/EIA-232, ANSI/EIA/TIA-232, etc. genannt werden. Die aktuelle Version lautet korrekt ANSI/EIA/TIA-232-F-1997, wobei das ‚F’ die Version, das Letzte die entsprechende Jahreszahl ist.

Die Datenübertragung erfolgt durch ein einfaches asynchrones, serielles Verfahren.

Seriell bedeutet, dass die einzelnen Bits des zu übertragenden Bytes nacheinander über eine einzige Datenleitung geschoben werden. Es gibt hier keine Kontrollleitung, die dem Kommunikationspartner mitteilt, wann das nächste Bit auf die Datenleitung geschickt wird. Sender und Empfänger müssen also mit exakt dem selben Takt arbeiten und der Sender muss dem Empfänger mitteilen, wann das erste Bit der Daten anfängt.

Alle RS232-Leitungen (mit Ausnahme der Masseleitung) arbeiten mit den Spannungspegeln im Bereich von +15V (für eine logische ‚0‘) und -15V (für eine logische ‚1‘). Der Bereich zwischen -3V und +3V ist undefiniert. Der Datenempfänger erwartet für eine 0 eine Spannung von über +3V und für eine 1 eine Spannung von unter -3V. Für den Sender gilt, dass der Spannungsbereich für den Ausgang mindestens +/-5V betragen muss, etabliert haben sich +/-12V. Für die Steuerleitungen (Handshakeleitungen) bedeutet ON einen hohen Pegel (+5V ... +15V) und OFF einen negativen Pegel (-5V ... -15V). Da sich diese Spannungspegel

GRUNDLAGEN

nicht besonders gut mit den heutzutage meist verwendeten TTL-Pegeln (Transistor-Transistor-Logik) vertragen, werden in der Regel spezielle Treiberschaltkreise eingesetzt, um diese Pegel auf TTL-Pegel anzupassen (‚1' = OFF = 5V; ‚0' = ON = 0V). Solche Treiberschaltkreise sind z. B. der MAX232 vom MAXIM. Die maximale Datenübertragungsrate liegt im Bereich von 50 bis zu 921.600 bps, wobei Geschwindigkeiten über 115.200 nicht mehr mit der herkömmlichen Schnittstellenhardware realisierbar sind. Die maximale Kabellänge ist im Bezug auf die Baudrate spezifiziert, bei 115.200 bps liegt die maximale Länge bei 1,5 Metern. Heutige Treiberbausteine sind jedoch bezüglich der elektrischen Leistung besser konstruiert, wodurch eine längeres Kabel sowohl theoretisch als auch praktisch möglich ist. Die genaue Länge ist aber Hardwareabhängig und kann nicht anhand der Datenblätter abgeleitet werden.

Das Übertragungsformat folgt einem festgelegten Muster. Zu Beginn wird zur Synchronisation das Startbit gesendet, gefolgt von 8 Datenbits (vom LSB zum MSB), dem Paritätsbit, einem Stoppbit und nach kurzer Ruhe ein weiteres Startbit um das Ende der Übertragung zu deklarieren. Das Paritätsbit dient der Fehlererkennung und gibt an, ob ursprünglich eine ungerade oder gerade Anzahl von „1"-Bits gesendet wurde. Es bezieht sich hierbei nur auf die Datenbits, die Start/Stoppbits werden nicht beachtet. Beispiel: das Datenbitmuster 0000001b soll übertragen werden, das Format ist 7E1 (sieben Datenbits, gerade Parität, ein Stoppbit). Das Paritätsbit für die Einstellung „Even" muss nun die „1" enthalten, damit die Gesamtanzahl Einsen gerade ist. Für das Bitmuster 0000011b ist das Paritätsbit nun auf „0" zu setzen. Im Allgemeinen Gebrauch werden meist folgende Einstellungen verwendet: 8 Datenbits, no Parity, 1 Stoppbit, in der Regel als 8N1 abgekürzt. Für die Übertragung eines Bytes sind damit mindestens 1+8 +1 „Bitdauern" nötig, woraus sich bei 115.200 Bit/s für die Nutzdaten ein Maximaldurchsatz von 92.160 Bit/s (=115.200*8/10 Bit/s) ergibt.

GRUNDLAGEN

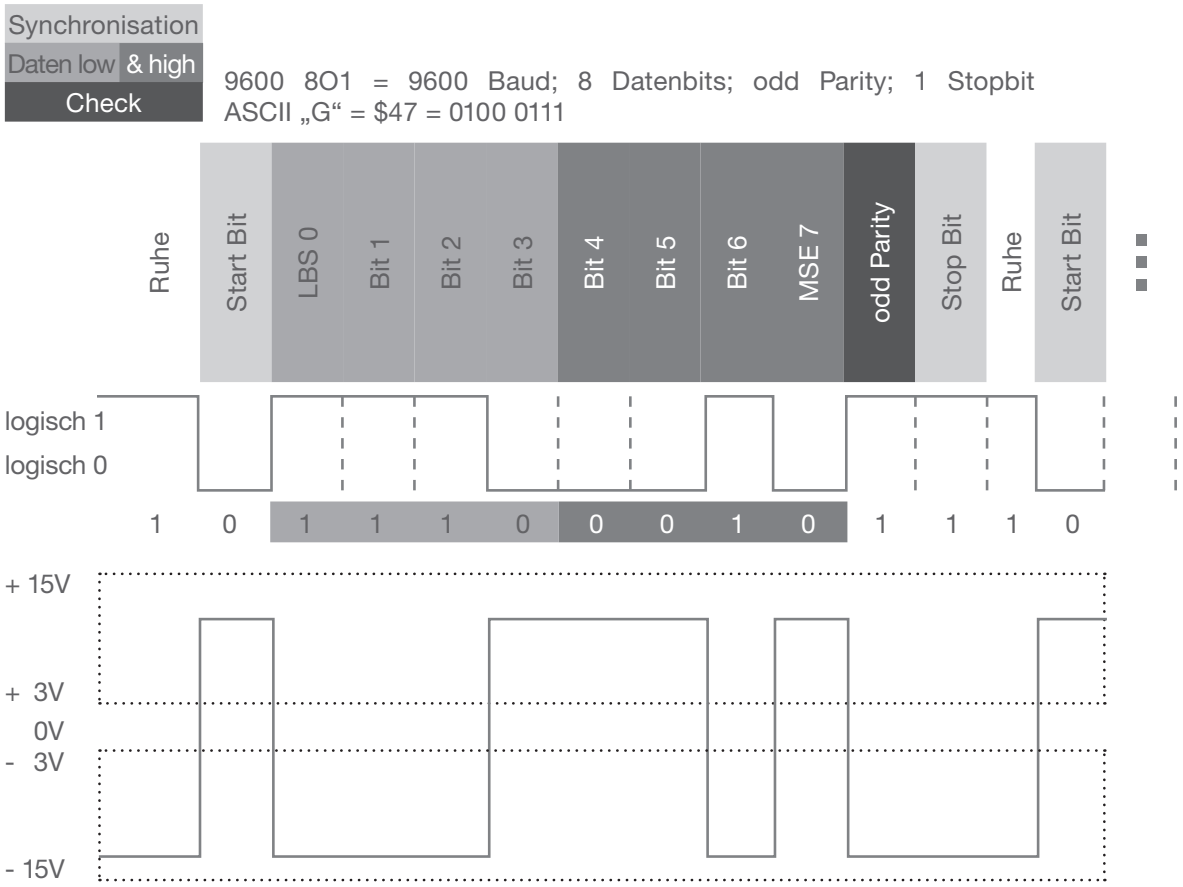

Abb 2.16 RS232 Datenrahmen

2.3.2 Software unter Windows [WINAPI]

Der Industriestandard EIA-232 legt nur den physikalischen Aufbau und den Datenrahmen fest, wie die Schnittstelle zu nutzen ist. Wie die Umsetzung des Protokolls genau realisiert wird kann von jedem Anwender/Entwickler selbst entschieden werden.

Die hier eingesetzte Bibliothek (als dynamic link library) wurde bereits von der Firma Stollmann E+V GmbH entwickelt und für dieses Projekt bereitgestellt. Die Konzeption dieser Bibliothek folgt in der Regel nach einem immer gleichen Muster, welches alle erhältlichen Applikationen mit Zugriff auf die serielle Schnittstelle leisten müssen. Da diese Bibliothek selbst zu definieren mit hohem Auf-

GRUNDLAGEN

wand verbunden und eine Erfolgsgarantie nicht gegeben ist, wird hier nur der grobe Funktionsablauf dargestellt.

Die nötigen Grundfunktionen um die Hardware anzusprechen werden bereits von der WindowsAPI (application programming interface, zu deutsch: „Schnittstelle zur Anwendungsprogrammierung“) bereitgestellt (enthalten in der windows.h, in dieser Form jedoch nur in W2K, XP, Vista).

- **HANDLE WINAPI CreateFile()**
  Hiermit wird das I/O-Objekt „ComPort“ erzeugt, der Rückgabewert ist ein Handle, mit welchem das erzeugte Objekt angesprochen werden kann. Übergabeparameter sind u. a. der Name des Objektes, die Funktionsart (read, write, both) und das sharing-attribut (soll hier nur ein einzelner oder mehrfacher Zugriff auch von anderen Applikationen erlaubt sein; für den ComPort sollte hier nur exklusiver Zugriff erlaubt sein).

- **BOOL WINAPI WriteFile()**
  Ausgabefunktion, leitet die übergebenen Daten an das ebenfalls übergebene Handle des I/O-Objektes weiter zur Ausgabe.

- **BOOL WINAPI ReadFile()**
  Leseprozedur, kann jederzeit ausgeführt werden, um eingehende Daten zu lesen. Übergabeparameter sind u. a. wieder das Handle des I/O-Objektes und ein Zwischenpuffer in welchem die Daten abgelegt werden sollen.

- **BOOL WINAPI DeleteFile()**
  Hier wird das I/O-Objekt geschlossen, die Hardware für neue Zugriffe freigegeben und eine andere Applikation kann ein CreateFile() auf die Hardware vornehmen.

- **BOOL WINAPI SetCommState()**
  Mit Hilfe dieser Methode können die Parameter eines Hardwaredevices geändert werden. In diesem Falle also die Parameter der seriellen Schnittstelle, das sind die Geschwindigkeit (Baudrate), Polarität sowie Anzahl Daten- und Stoppbits. Übergeben werden bei Aufruf das Handle von dem I/O-Objekt und ein Pointer auf eine Struktur, welche die nötigen Parameterangaben beinhaltet.

GRUNDLAGEN

Es gibt bereits eine fertige Struktur, die sogenannte DCB-Struktur, hier sind bereits alle nötigen Parametervariablen vorhanden und können leicht geändert werden.

Die erste Funktion liefert ein Handle auf das angesprochene Objekt. Generell gilt, dass mit diesen Funktionen nicht nur Hardwareperipherie (sämtliche Schnittstellen, Festplatten, etc.), sondern auch Dateien, namedpipes (Datenkanal der client/server Kommunikation), etc. angesprochen werden können. Der Rückgabewert ist hier entweder ein Handle oder die Fehlermeldung INVALID_HANDLE_VALUE. Alle restlichen Funktionen liefern einen boolschen Wert, entweder 0 für FALSE oder ungleich 0 für TRUE. Der genaue Wert im Erfolgsfalle ist nicht weiter spezifiziert. Für alle Funktionen ist das Schlüsselwort „WINAPI“ (Windows application programmer’s interface function) vorhanden. Es deklariert eine Funktion, die den Konventionen von Windows entsprechend übersetzt werden soll.

Die Bibliothek startet nach Erzeugen des ersten I/O-Objektes (hier der ComPort) einen Thread mit einer Endlosschleife. Dieser führt, wenn er nicht gerade Daten an die Schnittstelle weiterreicht, fortwährend die Funktion ReadFile() aus. Sobald neue Daten anliegen werden sie dadurch schnellstmöglich eingelesen und an die Hauptapplikation weitergeleitet. Dieses Verfahren wird durch das nachfolgend aufgeführte Sequenzdiagramm verdeutlicht.

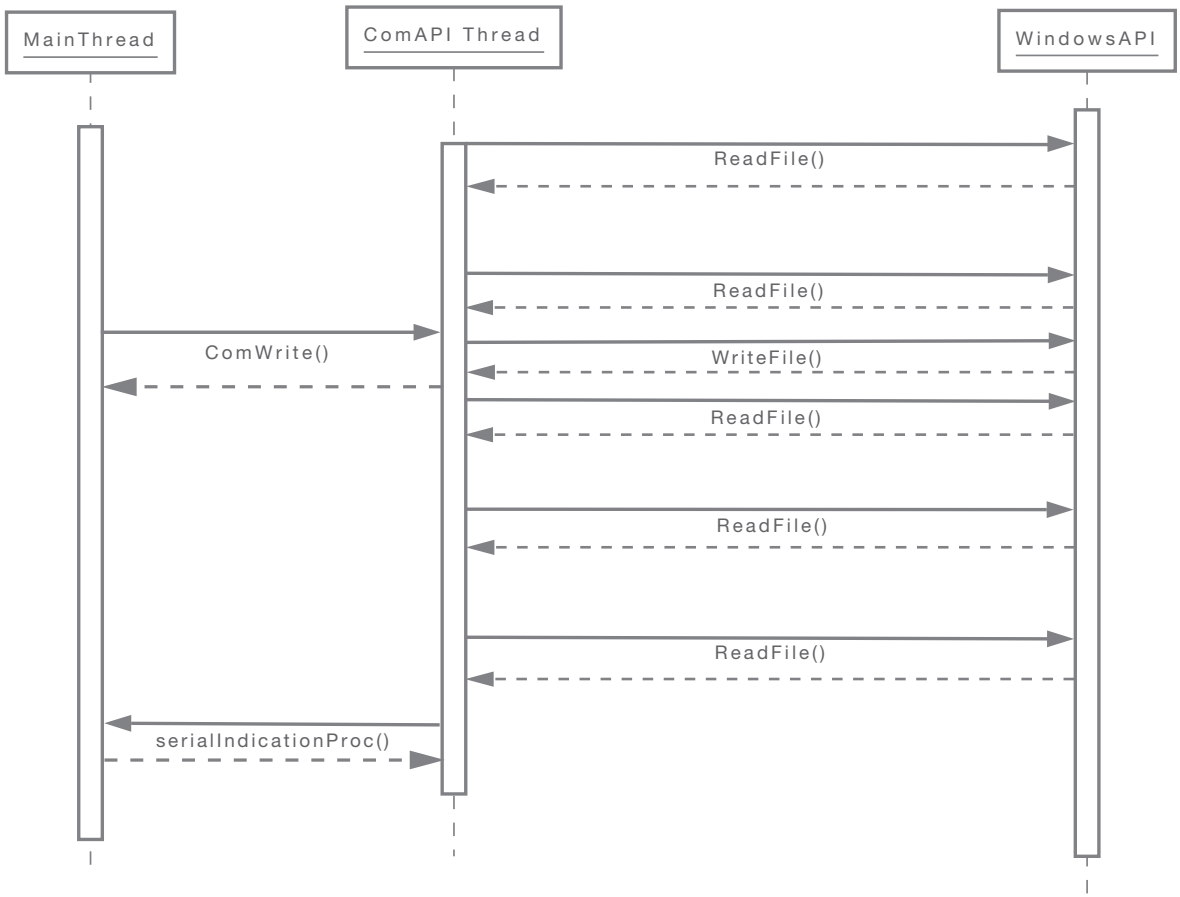

Abb. 2.17 Sequenzdiagramm ComAPI

Leider beinhaltet der ComAPI Thread aus Performancegründen keine Kollisionserkennung in der Form, dass er vor der Ausführung von Schreibzugriffen prüft, ob die serielle Schnittstelle mit dem Versenden von Daten bereits fertig ist ("Puffer leer?"). Ein verfrühter Aufruf überschreibt hierbei einfach alle noch im Sendepuffer vorhandenen Daten.

2.5 Die Stoppuhr (timewatch), [WINAPI]

Windows NT (ebenfalls der Rest der auf NT basierenden Windowsfamilie) besitzt eine einfache und gleichzeitig präzise Methode um Zeiten zu messen. Die Windows API stellt zwei Funktionen für den Systemtimer des Prozessors zur Verfügung (ebenfalls enthalten in der windows.h):

GRUNDLAGEN

*BOOL QueryPerformanceFrequency(LARGE_INTEGER *lpFrequency);*

Diese Funktion ermittelt die Anzahl von Zählschritten (ticks) pro Sekunde. Die CPU zählt pro Takt einen Counter hoch, dies stellt damit die präziseste Systemzeit dar. Die Anzahl der „ticks“ werden in der Variablen gespeichert, dessen Pointer übergeben wurde. Der Returnwert ist „0“, wenn die Methode fehlschlug, in diesem Fall steht für das System kein präziser Timer zur Verfügung. Im positiven Fall ist der Returnwert ungleich „0“, der genaue Wert ist nicht spezifiziert.

*BOOL QueryPerformanceCounter(LARGE_INTEGER *lpPerformanceCount);*

Diese Funktion ermittelt den aktuellen Wert des Counterregisters der CPU. Gespeichert wird dieser wiederum in der Variablen, dessen Pointer übergeben wurde. Der Returnwert folgt nach dem gleichen, oben beschriebenen Prinzip.

Die Funktion der Stoppuhr ist vergleichsweise simpel; die verstrichene Zeit zwischen Start und Stopp wird ermittelt, indem zweimal das Counterregister der CPU gelesen und gespeichert wird. Die Differenz dieser Werte ist die abgelaufene Zeit in ticks. Teilt man diesen Wert nun durch die ticks pro Sekunde, so erhält man die Anzahl der verstrichenen Sekunden. Diese kann man wiederum in Stunden, Minuten, Millisekunden, etc. umrechnen.

ANALYSE

3.1 Anforderungen an das Mess-System

Das fertige Zielsystem hat zur Aufgabe, Latenzzeiten, die in der Funkstrecke zwischen zwei Bluetoothmodulen entstehen, zu messen und Beeinflussungen durch das Betriebssystem zu berücksichtigen. Die Einhaltung von bestimmten Zeiten kann für einige Anwendungen nicht nur sinnvoll sondern auch notwendig sein. Auf Grund stärkerer Verbreitung und laufend sinkender Herstellungskosten der Bluetoothtechnologie versucht beispielsweise ein potentieller Kunde eine Anwendung, welche bisher über ein serielles Kabel verlief, nun mit Bluetooth zu realisieren. Hier hat der Kunde eventuell bestimmte Übertragungszeiten realisiert, mit denen er für seine Produkte wirbt. Um diesen Standard einzuhalten muss der Kunde zunächst ein Modul finden, welches seine gesetzten Rahmendaten erfüllt. Ein geeignetes Mess-System muss hier die notwendigen Daten liefern, schließlich kann nicht pauschal gesagt werden welcher Chip welche Zeiten einhält. Dies hängt neben der Chiparchitektur auch mit der verwendeten Software zusammen. Insofern können die Übertragungszeiten zweier baugleicher Module mit unterschiedlicher Software voneinander abweichen, teilweise auch nur situationsbedingt. Um das spezielle Verhalten eines Bluetoothmoduls herauszufinden, ist eine Messumgebung unverzichtbar. Der Aufbau eines Mess-Systems mit diversen, präzisen elektronischen Messgeräten ist jedoch meist sehr zeitaufwendig und um eine grobe Einschätzung des IST-Zustandes zu erhalten nicht geeignet. Nicht immer muss der Entwickler den genauen Messwert wissen, eine ungefähre Angabe reicht oft aus. Teilweise ist es auch nicht möglich diese Werte zu ermitteln ohne die entsprechenden Geräte zu öffnen um an die Signalleitungen zu gelangen. Das zu erzeugende System soll einem Entwickler mit wenigen Handgriffen die Möglichkeit einer Messung ohne Messinstrumente – auf Basis der seriellen Schnittstelle – schaffen.

ANALYSE

3.2 Testszenarien und Aufbau

Folgender Aufbau der Testumgebung soll realisiert werden:

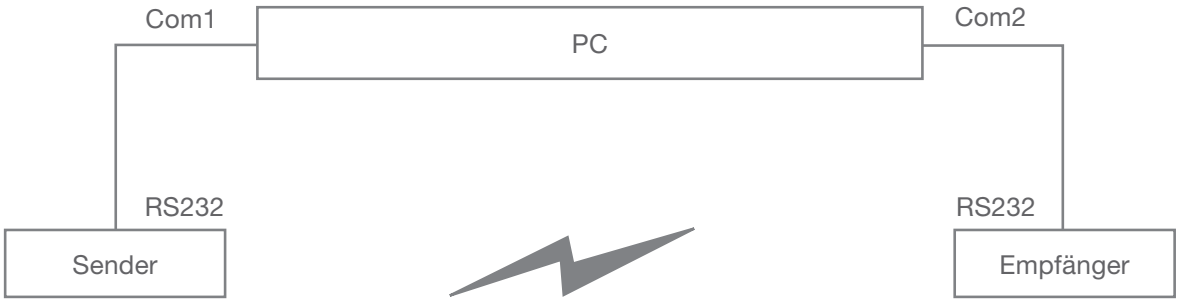

Abb. 3.1 Messaufbau

Der PC soll Daten aus dem ASCII-Zeichensatz über eine serielle Schnittstelle senden und auf einer Zweiten die Daten zurückerwarten, wobei die hier verstrichene Zeit protokolliert wird. Der Weg verläuft dabei über zwei Bluetoothmodule welche mittels Funk verbunden sind. Die maximale Entfernung der Module ist durch die Kabellänge begrenzt. Der Datenaustausch soll in beide Richtungen erfolgen können, wobei die Länge der Datenpakete (1 bis n Zeichen) frei definierbar sei. Es gilt hierbei auch zu testen, ob die Entfernung zwischen den Funkmodulen eine entscheidende Rolle spielt oder vernachlässigbar ist.

3.3 Funktionale Anforderungen

Das Programm baut über die Bluetoothgeräte eine virtuelle, serielle Verbindung auf. Diese wird mittels des vorhandenen SPP-Protokolls durchgeführt. Die Software soll den Verbindungsaufbau komplett selbst bewerkstelligen, die hierfür benötigten Bluetoothgeräteadressen müssen variabel einstellbar sein. Als Nutzdaten sind hier variable Datenblöcke vorgesehen mit einer frei skalierbaren Größe; der Anwender soll also jederzeit entscheiden können, wie viele Zeichen er versenden will. Die Auswertung der ermittelten Messwerte muss entweder sofort oder nach Aufforderung ausgegeben werden. Das Programm benötigt Prozeduren um Zeiten zu messen und um auf die serielle Schnittstelle zuzugreifen. Auf Grund

ANALYSE

der durch das Windowssystem erzeugten Latenzzeiten, die durch das ansprechen der seriellen Schnittstelle entstehen, muss eine Einmessprozedur realisiert werden. Bedingt durch das in Kapitel 2.1.4 beschriebene Verhalten des Schedulers reicht die Ermittlung eines einzigen Wertes nicht aus. Zwar ist es möglich die Applikation mit der Priorität „Realtime“ zu starten um Unterbrechungen durch weitere Prozesse zu unterbinden und stabilere Verzögerungszeiten zu erlangen, spätere Tests haben jedoch ergeben, dass dieser Sachverhalt keinerlei Einfluss auf die danach ermittelten Verzögerungszeiten hat. Diese wichen in allen Prioritätsstufen voneinander ab. Grund hierfür ist einerseits, dass Windows keine richtige Echtzeit bieten kann und andererseits, dass die Prozessorzeit jedes Prozesses durch die „Quantum“ Zeit begrenzt ist und damit jeder Prozess jederzeit in seiner Ausführung unterbrochen werden kann. Die Einmessprozedur muss daher eine durchschnittliche Verzögerung ermitteln und in die Auswertung mit einbeziehen.

Auf Grund der bereits in Kapitel 2.3 erwähnten Beschränkung bezüglich der Kabellänge in Verbindung mit der Baudrate kann für den Messaufbau zunächst nur eine Kabellänge von 1,5 Meter für beide Bluetoothmodule berücksichtigt werden um den Abstand zwischen den Modulen für eine längere Funkstrecke zu erweitern. Tests haben zwar ergeben, dass eine Verdoppelung der Kabellänge bei den hier verwendeten Geräten keine negativen Auswirkungen hatte, jedoch ist dieses Verhalten wie bereits erwähnt nicht garantiert und nur von der Hardwarequalität abhängig. Es wäre zwar möglich die Baudrate zu verringern um eine größere Kabellänge und damit einen größeren Abstand zwischen den Funkmodulen zu erreichen, jedoch vergrößern sich hier die Latenzzeiten enorm.

3.4 Nichtfunktionale Anforderungen

Auf Grund der hohen Verbreitung von Windowssystemen, kann man davon ausgehen, dass der Arbeitsplatz des Entwicklers in der Regel nicht mit einem Echtzeitsystem oder proprietärem System ausgestattet ist, sondern eher ein windowsbasierendes System anzutreffen ist. Wie bereits in Kapitel 2.1 erwähnt, hat der Programmierer hier jedoch keinen direkten Zugriff auf die installierten Hardwareperipherien, dieser wird durch den Windowskernel aus Sicherheits-

ANALYSE

gründen verhindert. Es muss sich also einer fertigen Bibliothek bedient werden, um überhaupt Zugang zur seriellen Com Schnittstelle zu erlangen. Nachteil hierbei ist jedoch, dass der Zugriff immer betriebssystemvermittelt erfolgt. Die Zugriffsanforderung wird also zunächst vom user mode an den kernel mode weitergereicht, wo die bereits erwähnte Ablaufprozedur durchlaufen wird, also Aufruf Local Procedure Call Facility, I/O Manager, Object Manager und Security Reference Monitor. Letztendlich entscheidet der Scheduler im Mikrokernel, wann welcher Prozess zum Zuge kommt. Durch diese Faktoren kann von einer zeitlich gesicherten Umgebung nicht gesprochen werden. Eigene Messungen haben jedoch ergeben, dass die Verzögerung durch das Windowssystem zwar variieren, aber im Schnitt für jeweils eine bestimmte Anzahl Zeichen gleich sind. Die Abweichungen betragen in der Regel zwischen 0 bis 5ms über und unter dem Mittelwert. Demnach ist es sinnlos, das System mit einer Genauigkeit im µs-Bereich anzulegen, dieses System soll jedoch auch nicht dem Anspruch eines Echtzeitsystems genügen sondern eine schnelle und unkomplizierte Messung ermöglichen. Trotz allem sollte das Windowssystem so wenig parallele Prozesse wie möglich aktiviert haben (z. B.: Mailprogramm, Virenscanner, etc.). Daher wird auch auf den Einsatz einer grafischen Windows-GUI verzichtet. Das Programm wird daher als Konsolenapplikation ohne unnötige Prozeduren laufen.

Der Verbindungsaufbau der Geräte sollte einfach und schnell zu bewerkstelligen sein, eine lange Einarbeitung soll vermieden werden, um nur eine kurze Messung durchzuführen. Generell sollte sich das System nach außen hin so einfach wie möglich darstellen, der Anwender soll sich nicht mit Umgebungsvariablen oder einer langwierigen Installation auseinandersetzen müssen. Die einzige Einstellung, die für eine Messung notwendig wird, soll lediglich das Setzen der Bluetoothadresse des Slaves sein.

DESIGN

Für die Umsetzung des Systems gibt es zwei Möglichkeiten. Zum Einen wäre die Realisierung als proprietäres System denkbar, wo die Bluetoothmodule an eine dem Entwickler inhaltlich unbekannte Box angeschlossen werden. In dieser Box könnte ein kleiner Mikroprozessor mit kleinem Display für die Messwertausgabe und Bedieninterface vorhanden sein. Als Betriebssystem könnte hier QNX, MS-DOS, etc. dienen. Der Vorteil hierbei wäre zwar für das System vorhersehbare Beeinflussungen der Latenzzeiten, jedoch muss dieses Gerät stets verfügbar sein und extra entwickelt werden. Die Zweite Möglichkeit, welche in dieser Arbeit auch verfolgt wird, ist die Realisierung mit einem handelsüblichen PC. Diese wirft zwar einige zu lösende Probleme bezüglich schwankender Latenzen im Betriebssystem auf, jedoch ist hierbei der Vorteil gegeben, dass das System leicht auf allen PC-kompatiblen Geräten unter Windows ausführbar ist. So kann beispielsweise ein Entwickler mittels Laptop jederzeit und räumlich unbegrenzt eine schnelle Messung auch beim Kunden, etc. durchführen. Die Hardware (PC) sollte in jedem IT-Unternehmen vorhanden sein und müsste somit nicht extra angefertigt werden.

Die Baudrate auf der seriellen Schnittstelle wird auf 115.200 bps festgesetzt. Dies ermöglicht eine möglichst kleine Latenzzeit bei der maximalen für die Schnittstelle spezifizierten Geschwindigkeit, ohne bauliche Veränderungen vorzunehmen. Eine serielle Schnittstelle nach RS232 kann zwar eine maximale Geschwindigkeit von 921.600 bps leisten, was jedoch nur unter Einsatz von USB-to-RS232 Konvertern möglich ist. Konverter von USB auf RS232 sind zwar weit verbreitet erhältlich, erfahrungsgemäß ist die problemlose Funktionstüchtigkeit nicht immer gegeben und es ist nicht bekannt ob und inwieweit diese Konverter den Datenverkehr beeinflussen. Daher wird der Einsatz selbiger hier zunächst vernachlässigt.

Auf eine komplette Testautomatisierung muss vorerst verzichtet werden. Grund hierfür sind erstens entstehende Messfehler sobald der Sendeaufruf wiederholt in einer Schleife erfolgt. Bereits bei der Realisierung der Einmessprozedur sind hier Probleme aufgetaucht, so dass der wiederholte Aufruf in einer Schleife immer den gleichen Latenzwert auch bei verschiedenen Systemlasten aufwies, was nachweislich falsch ist. Selbst eine kurze Pause mittels des Aufrufs von „sleep()“ verschaffte keine Abhilfe. Eine leere for-Schleife, die von 0 bis 100.000.000 zählt schaffte zwar für die Einmessprozedur Abhilfe, für die eigentlichen Messungen

DESIGN

wurde hierauf jedoch nach längerer Überlegung verzichtet. Dies ermöglicht es einem Entwickler nun jederzeit einen zeitlich dynamischen Abstand zwischen die Messungen zu setzen, um beispielweise auf sich verändernde Umgebungsvariablen (Störungen der Funkstrecke durch Handy, Barriere, etc.) zu reagieren oder auch problemlos den Messaufbau zwischenzeitlich zu verändern. Zweitens kann eine komplette Testautomatisierung, also das Testen aller Senderichtungen mit stetig steigenden Datenlängen, nicht realisiert werden, da die durchschnittliche Verzögerungszeit bereits unter Windows nicht konstant ist. Diese steigt proportional zur Datenlänge an, das genaue Verhalten wird im folgenden Kapitel 5 (Tests) näher beschrieben.

Die Software besteht schließlich aus folgenden wesentlichen Teilen:

1. Bibliothek für den Zugriff auf den ComPort
2. Die Stoppuhr
3. Die Einmessprozedur
4. Eingabefenster und Auswertung der Daten

Diese Teile werden in den später folgenden Abschnitten detailliert beschrieben. Das UML-Diagramm auf der nachfolgenden Seite zeigt zunächst die Programmstruktur des Mess-Systems.

Der Aufbau ist schlicht, die Datei „messsystem.cpp“ ist der zentrale Punkt, hier enthalten ist die main() und alle weiteren Funktionen. Die Hilfsfunktionen sind in andere Teile ausgelagert, welche letztendlich auf der WindowsAPI aufbauen.

DESIGN

Abb. 4.1 Einfaches Klassendiagramm mit den nötigsten Methoden

DESIGN

Die Wahl der Entwicklungsumgebung fiel auf Microsoft Visual Studio 6.0, da sich hier die Einbindung der bereits vorhandenen API und Bibliotheken recht einfach gestaltete. Die Umsetzung erfolgte in der Programmiersprache C. Vorgestellt werden alle wesentlichen Funktionen inklusive der Übergabe- und Rückgabeparameter. Sofern es sinnvoll erschien, wurde ein Beispiel für den jeweiligen Methodenaufruf aufgeführt. Ein Sequenzdiagramm beschreibt eine reguläre Standardsituation schrittweise.

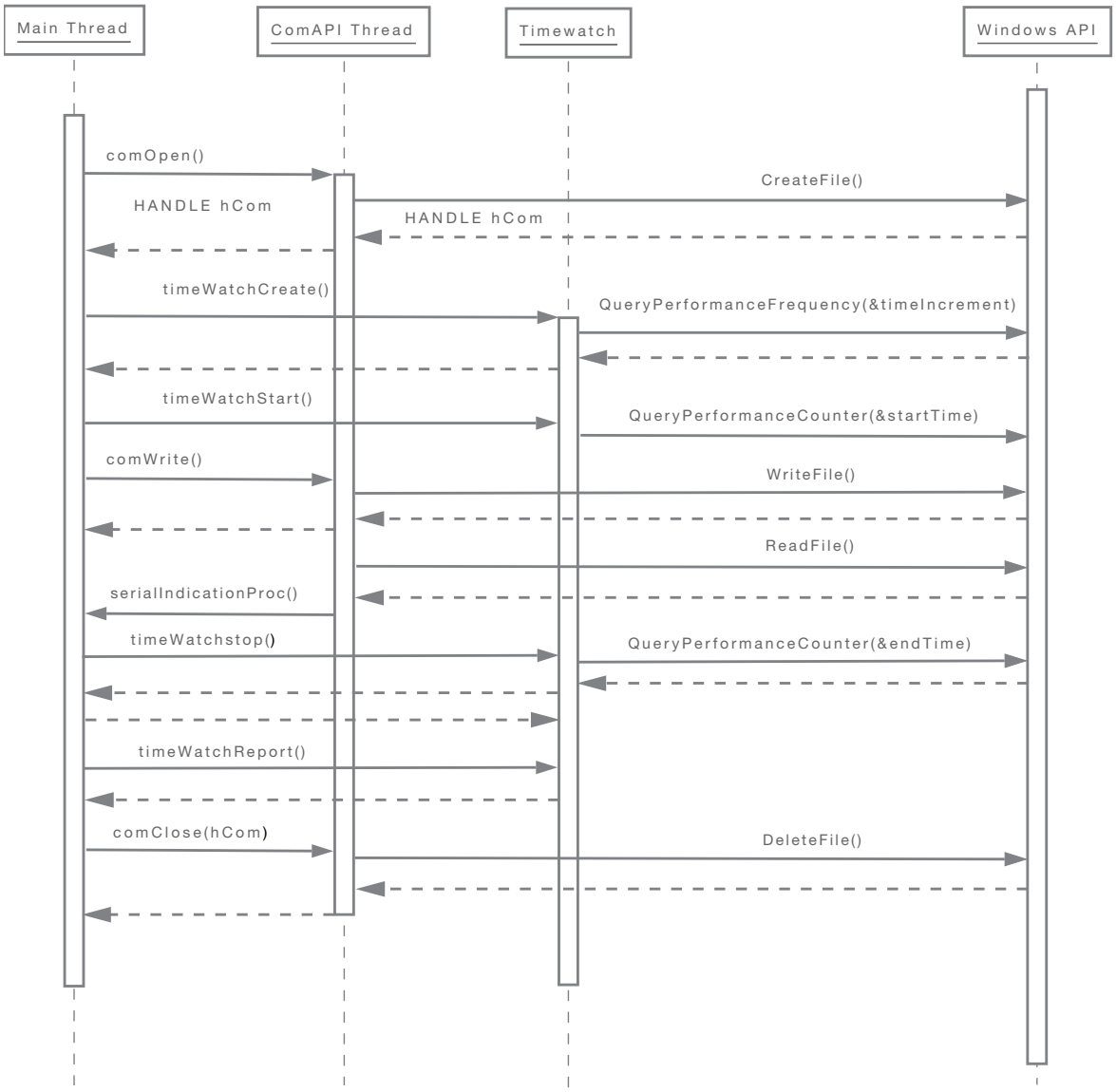

Abb. 4.2 Sequenzdiagramm, beschreibt den grundsätzlichen Programmablauf

DESIGN

4.1 Softwarefunktionen

4.1.1 Bibliothek für den Zugriff auf den ComPort

Die Bibliothek in ihrer Grundfunktion und Arbeitsweise wurde bereits in einem vorangegangenen Kapitel beschrieben, hier werden die nötigen Funktionen aufgezeigt.

Name	comOpen()
Übergabeparameter	PCHAR pszDeviceName, PCHAR pszParameter, PComCallBack callBack, PVOID callBackContext, HCOM * phCom
Returnwert	Handle auf das I/O-Objekt
Beschreibung	Öffnet die serielle Schnittstelle für die Kommunikation, übergeben werden Name, Geschwindigkeit, Methode für events und den Namen für das Handle.

Bsp.: comOpen(„COM1“,“Baudrate=115200“,serialIndicationProc, NULL, &hCom1);

Name	comWrite()
Übergabeparameter	HCOM hCom, PBYTE pvBuffer, UINT uBufferLength
Returnwert	INT
Beschreibung	Sendet eine Zeichenkette vom Typ PBYTE mit einer zu definierenden Länge an den Handle des I/O-Objekts. Die Länge muss nicht mit der Länge der Zeichenkette übereinstimmen, es werden nur maximal Anzahl Länge Zeichen ausgegeben. (Vorsicht bei Länge > pvBuffer.size!!!)

Bsp.: comWrite(hCom1, (PBYTE) „1“,1);

DESIGN

Name	comClose()
Übergabeparameter	HCOM hCom
Returnwert	INT
Beschreibung	Schließt das dem Handle zugehörige I/O-Objekt.

Bsp.: comClose(hCom1);

Die letzte vorzustellende Methode ist der Eventhandler der seriellen Schnittstelle. Dieser ist nur indirekt ein Teil der Bibliothek, lediglich das Vorhandensein wird vorausgesetzt. Die funktionalen Inhalte können vom Entwickler selbst definiert werden. Es werden die von der seriellen Schnittstelle zurückgereichten Events ausgewertet und dazu passende Aktionen ausgeführt (z. B.: stoppen der Stoppuhr). Der Aufruf selbst erfolgt durch den Thread, der auch für die Aufrufe der Read-/WriteFile() Funktionen zuständig ist.

Name	serialIndicationProc()
Übergabeparameter	PVOID pvContext, TComEvent event, UINT ulStatus, PVOID pvData, UINT ulLength
Returnwert	void
Beschreibung	Eventhandler für die serielle Schnittstelle, wird aufgerufen, wenn ein Event vom serial Controller Objekt aus der Bibliothek vorliegt. Events sind hier erfolgreiches Öffnen/Schließen der Schnittstelle und eingehende Daten.

DESIGN

4.1.2 Die Stoppuhr

Die Funktion wurde bereits in Kapitel 2.5 beschrieben, es wird die Differenz zwischen zwei Zeitpunkten errechnet. Der Startzeitpunkt ist direkt vor dem Senden der Daten, der Stoppzeitpunkt erfolgt direkt im Aufruf der Methode, die die Events für den ComPort verarbeitet. Da die Latenzzeiten nie gleich sind, wird hier fortlaufend der Durchschnitt ermittelt. Der Anwender bestimmt selber, wie viele Messvorgänge durchgeführt werden durch jeweils erneutes Ausführen der entsprechenden Funktion.

Name	TimeWatchCreate()
Übergabeparameter	int index PCHAR pszName
Returnwert	BOOL
Beschreibung	Erzeugt ein neues Stoppuhrenobjekt, welches über den gewählten Index ansprechbar ist. Der Index ist der Platz im Stoppuhrencontainer, der Name ist frei wählbar.

Bsp.: timeWatchCreate(TW_SLOT_1, „Timer 1“);

Name	TimeWatchStart()
Übergabeparameter	int index
Returnwert	void
Beschreibung	Speichert die aktuelle Prozessorzeit im durch den Index identifizierten Objekt (startTime).

Bsp.: timeWatchStart(TW_SLOT_1);

DESIGN

Name	TimeWatchStop()
Übergabeparameter	int index
Returnwert	void
Beschreibung	Speichert die aktuelle Prozessorzeit im durch den Index identifizierten Objekt (endTime).

Bsp.: timeWatchStop(TW_SLOT_1);

Name	TimeWatchReport()
Übergabeparameter	void
Returnwert	void
Beschreibung	Gibt den letzten aktuellen Verzögerungswert auf der Konsole aus.

Bsp.: timeWatchReport();

Name	getAWDT()
Übergabeparameter	void
Returnwert	INT
Beschreibung	Dient zur Abholung des letzten Verzögerungswertes zur Berechnung des Durchschnitts.

Bsp.: adt += getAWDT();

DESIGN

Funktionen der Headerdatei:

Name	reset()
Übergabeparameter	void
Returnwert	void
Beschreibung	Wird zu Beginn gestartet, ermittelt mit „QueryPerformanceFrequency(&timeIncrement)“ die aktuelle Prozessorfrequenz.

Name	start()
Übergabeparameter	void
Returnwert	void
Beschreibung	Ermittelt mit „QueryPerformanceCounter (&startTime)“ den aktuellen Wert des Counterregisters von der CPU, speichert in der Variablen „startTime“.

Name	stop()
Übergabeparameter	void
Returnwert	void
Beschreibung	Ermittelt mit „QueryPerformanceCounter (&endTime)“ den aktuellen Wert des Counterregisters von der CPU, speichert in der Variablen „endTime“.

DESIGN

Name	get()
Übergabeparameter	void
Returnwert	ULONG
Beschreibung	Ermittelt die Differenz zwischen „startTime“ und „endTime“ und rechnet diese dann von Sekunden in ms um.

4.1.3 Die Einmessprozedur

Direkt nach dem Programmstart wird das System einer Einmessprozedur mit der erforderlichen Mindestdatenlänge von 1 Zeichen unterworfen. Hierbei soll der kleinste durchschnittliche Verzögerungswert ermittelt werden, der durch das Windowssystem zustande kommt, sobald Daten an die serielle Schnittstelle gesendet werden. Dies kann auf zwei Wegen erfolgen:

1. Am Ende eines Kabels werden die Leitungen RX und TX mittels geeignetem Stecker (sogenannter Loopbackstecker) verbunden.

2. Beide ComPorts werden durch den Zusammenschluss beider Kabel mit einem Nullmodemadapter verbunden.

Beide Wege leisten das selbe Ergebnis, es wird die Round-Trip-Time, also die Zeit, die auf dem Weg von der Applikationsebene bis zum Kabelende und zurück abläuft, gemessen. Mehrere Messdurchläufe ergeben dann die Durchschnittszeit. Diese wird später bei der Auswertung von der Gesamtverzögerungszeit des gesamten Systems abgezogen um die reine Latenzzeit der Bluetoothdevices inklusive der Funkstrecke zu ermitteln. Die Ermittlung der Durchschnittsverzögerungszeit kann jederzeit wiederholt werden, auch mit der jeweils aktuellen Datenlänge (1 bis 200 Zeichen). Auf Grund von Messfehlern musste zwischen den einzelnen Messungen eine Verzögerung in Form einer Warteschleife eingesetzt werden, ohne diese würde der Messwert immer 20ms betragen. Der Grund hierfür war bisher nicht ermittelbar, vermutlich spielt hier der Scheduler eine Rolle, der die Prioritäten für die laufenden Prozesse dynamisch vergeben und ändern kann.

DESIGN

Der Aufruf der Funktion setTime() erfolgt einmalig zum Start, später auch aus dem Menü, danach rekursiv durch den Eventhandler (serialIndicationProc()) der seriellen Schnittstelle. Die Anzahl der rekursiven Aufrufe ist durch eine Variable begrenzt (momentan 20 Wiederholungen).

Name	setTime()
Übergabeparameter	void
Returnwert	void
Beschreibung	Ermittelt die Verzögerungszeit, die durch das Windowssystem erzeugt wird.

4.1.4 Eingabefenster und Auswertung der Daten

Das Hauptprogramm wurde, wie bereits erwähnt, als Konsolenapplikation realisiert. Die kleine TextGUI sieht so aus:

Bluetoothadresse wechseln(0)
Messung starten M->S Block(1)
Messung starten S->M Block(2)
System anlernen (3)
Messung starten M->S Byte(4)
Messung starten S->M Byte(5)
Bluetooth SPP aufbauen (6)
Reset (7)
Blocklaenge aendern (8)
Beenden (9)
Menue (?)
Eingabe:

Die Bezeichner „S“ und „M“ stehen hier für „Slave“ und „Master“. Unter der Option 0 kann die Bluetoothadresse des Slaves neu gesetzt werden. Für die Testzwecke ist hier bereits eine Defaultadresse eingesetzt, da aber die Software auch für alle Devices nutzbar sein soll kann diese wie gesagt neu beschrieben werden. Die Optionen 1 und 2 übertragen einen Zeichenblock der vorher defi-

DESIGN

nierten Länge (default 1), während bei 4 und 5 immer genau 1 Zeichen übertragen wird. Option 3 ruft die Funktion setTime() auf um das System neu anzulernen, die aktuell unter der Option 8 eingestellte Länge ist hierbei zu beachten. Dies war auf Grund steigender Windowslatenzzeiten in Verbindung mit einer steigenden Blocklänge nötig. Unter Punkt 6 wird eine neue Bluetooth SPP Verbindung aufgebaut, dies erfolgt mittels des AT-Befehls „atd{Bluetoothsadresse}“. Dieser Befehl sollte auch für andere Geräte funktionieren, die mittels AT-Befehlssatz konfiguriert werden können. Im Falle von Option 7 wird ein Reset des Systems durchgeführt, dabei werden alle ermittelten Werte, außer der Durchschnittslatenzzeit des Windowssystems, gelöscht und die Bluetoothverbindung getrennt. Vorher werden die aktuellen, bis dato ermittelten Durchschnittswerte ausgegeben. Punkt 8 ändert wie oben angegeben die aktuelle Länge des Zeichenblocks auf 1 bis 200 Zeichen. Weitere Längenangaben sind momentan nicht erlaubt, wären jedoch theoretisch ohne Weiteres möglich. Option 9 schließt die seriellen Ports und beendet das Programm. Durch Eingabe des „?“ wird das Menü ausgegeben, dies war zur besseren Übersicht sinnvoller anstatt die Ausgabe nach jeder Aktion zu wiederholen.

TEST

In dieser Sektion wird die Software einigen anwendungsbezogenen Tests unterzogen. Die Hardware ist soweit bekannt, die verwendeten Komponenten werden Zwecks der Vollständigkeit nochmals aufgeführt:

- Windows PC mit Windows 2000 (NT basierend)
- 2 Entwicklerboards (BlueEva+ C11/G2) mit jeweils einem montierten Bluetoothmodul (Class 1) und serieller RS232-Schnittstelle
- 2 serielle Verbindungskabel, Länge 1,5 Meter
- Loopbackstecker für die Einmessprozedur
- Kabelverlängerung für die seriellen Kabel um die Luftlinie zwischen den Modulen zu vergrößern

5.1 Messaufbau

Die zu testenden Entwicklerboards werden mit den Kabeln direkt an die seriellen Schnittstellen (ComPorts) an Com1 und Com2 angeschlossen.

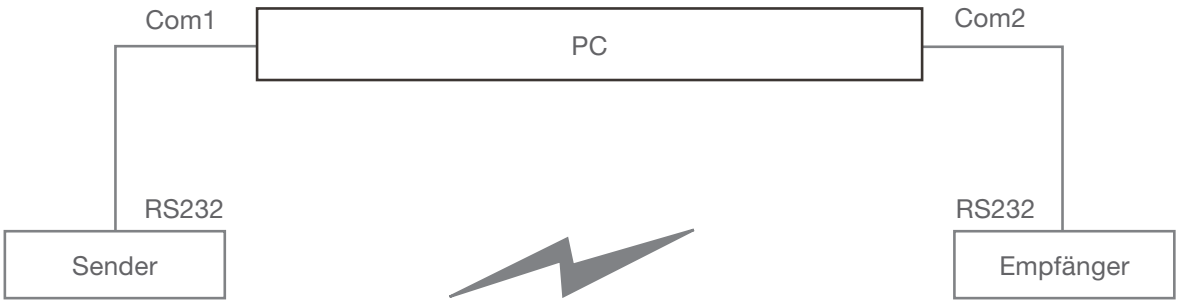

Abb. 5.1 Aufbauplan und Anschluss-Skizzee

5.2 Durchführung

Vor dem Programmstart muss zunächst das Modul am Com1 mit dem Loopbackstecker getauscht werden. Das Programm kann nun aus dem normalen Windowsbetrieb gestartet werden. Nach erfolgreicher Durchführung der Einmessprozedur erscheint das Menü, der Loopbackstecker muss nun wieder mit dem vorher getrennten Modul ausgewechselt werden. Für die Testoption „Block“

TEST

muss schließlich noch eine gewünschte Blocklänge zwischen 1 und 200 Zeichen eingestellt werden. Zu beachten ist jedoch, dass bei einer Länge >20 Zeichen die Ergebnisse verfälscht sind. Das System wird zu Beginn nur mit einer Länge von einem Zeichen angelernt. Tests haben ergeben, dass bei sich ändernder Blocklänge die Latenzzeiten innerhalb des Windowssystems steigen (average Windowsdelaytime AWDT, siehe nachfolgende Grafik). Für korrekte Messergebnisse ist ein erneutes Anlernen des Systems nach Änderung der Blocklänge unverzichtbar. Um dies zu verdeutlichen wurden die Verzögerungswerte des Windowssystems jeweils erfasst und grafisch dargestellt:

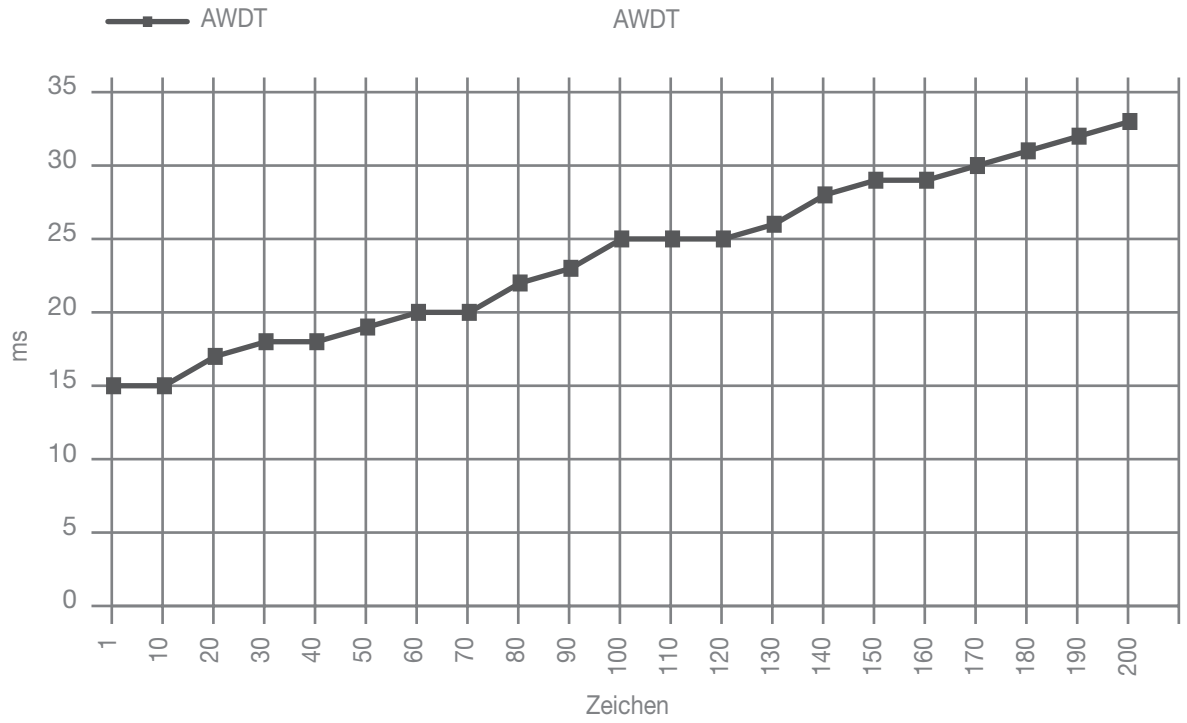

Abb. 5.2 Average Windows Delay time bezogen auf die Blockgröße

Sind soweit alle vorangegangenen Einstellungen korrekt, kann nun die Bluetooth SPP Verbindung zwischen den beiden Modulen aufgebaut werden. Der Erfolg dieses Aufbaus wird durch die Meldung „CONNECTED!“ auf der Konsole quittiert, andernfalls erfolgt die Meldung „NO CONNECTION!“. Der Anwender kann nun über die Menüoptionen die Latenzzeit ausgehend vom Master oder vom Slave mit den Nutzdaten „Byte“ (1 Zeichen) oder „Block“ (n Zeichen) ermitteln.

TEST

Jede Ausführung einer einzelnen Messung ermittelt jeweils einen neuen Wert, aus allen wird fortlaufend der Durchschnitt gebildet.

5.3 Messergebnisse

Die erste Messung wurde mit einer Zeichenlänge von 1 bis 200 Zeichen durchgeführt, der Abstand zwischen den Modulen betrug ca. 30 cm. Jeweils 15 Einzelmessungen mit steigender Blocklänge in Zehnerschritten wurden als Grundlage für das folgende Diagramm durchgeführt. Die hier angezeigten Latenzzeiten sind bereits ohne die oben aufgeführten Verzögerungen des Windowssystems dargestellt. Da die Bluetoothtechnologie nicht festgelegten Latenzzeiten unterliegt schwanken die Verzögerungszeiten leicht. Anhand des Diagramms ist trotzdem leicht zu erkennen, dass die Latenzzeit proportional zur Datenlänge steigt.

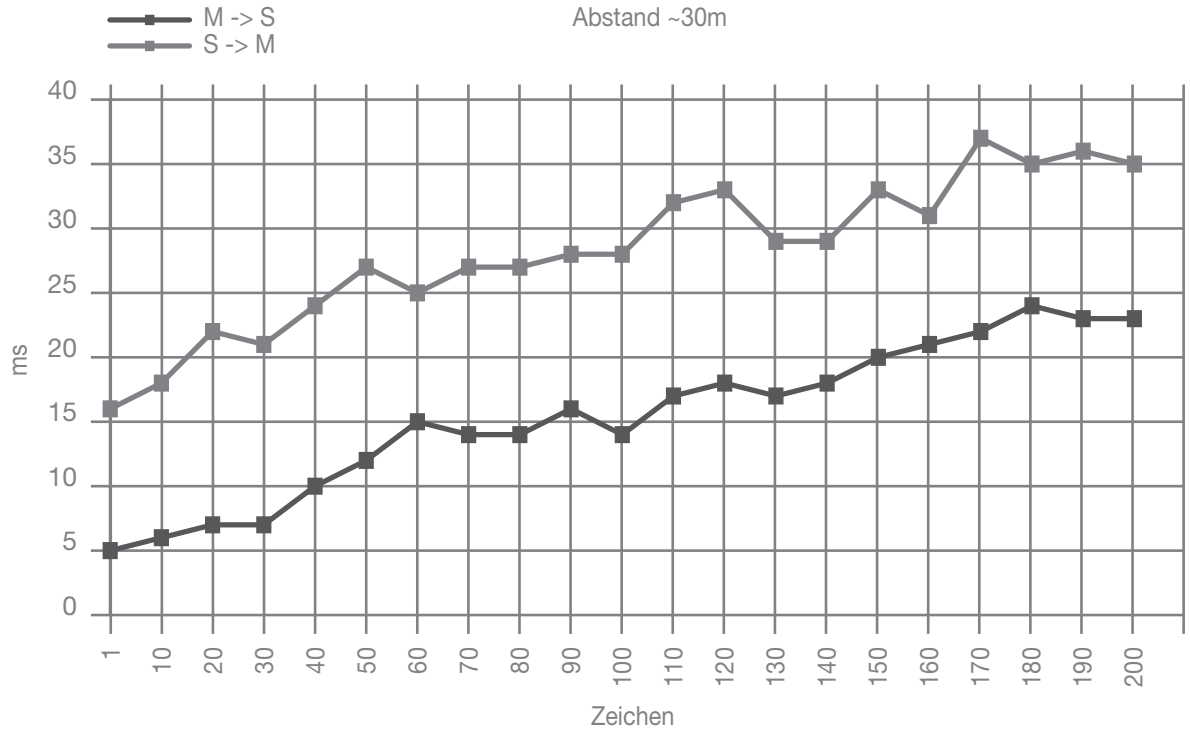

Abb. 5.3 Latenzzeit in beiden Richtungen, Abstand ca. 30cm

Weitaus gravierender zeigt sich allerdings der Sachverhalt, dass die Verbindung zwischen den Modulen nicht gleich zu sein scheint. Der Datentransfer vom Slave

TEST

zum Master erscheint hier generell langsamer zu sein als umgekehrt. Dieses Verhalten ist hingegen nicht auf einen Defekt sondern auf eine plausible Erklärung zurückzuführen. In einer der untersten Schichten, dem Baseband, ist eine solche Verzögerung spezifiziert [BT4]. Wie bereits bekannt, kann der Master mit bis zu sieben Slaves in einem Pikonetz kommunizieren. Um diese Slaves gleichermaßen bedienen zu können bedarf es hier eines geregelten Algorithmus. In diesem fragt (poll) der Master jeden Slave der Reihe nach an, ob Nutzdaten übertragen werden sollen und übergibt in dem Pollpaket auch den Slot für die Antwort (slave-to-master-slot). Der Slave, welcher immer – wie alle anderen teilnehmenden Slaves auch – auf dem master-to-slave-slot eingehende Pakete annimmt, darf nur in dem für ihn bestimmten slave-to-master-slot eine Antwort geben. Hat der Slave keine Nutzdaten an den Master zu senden, so wird zumindest ein Null-Paket gesendet damit der Master nicht von einem Übertragungsfehler ausgehen muss. Sind Daten vorhanden, werden sie in dem für den Slave reservierten Slot gesendet. Der Master gibt hierbei auch die Länge des Paketes vor (1-, 3- oder 5-Slotpaket). Dieses Prinzip dient zum Einen der gleichmäßigen Behandlung der teilnehmenden Slaves durch den Master, zum Anderen zur Verhinderung von Datenkollisionen.

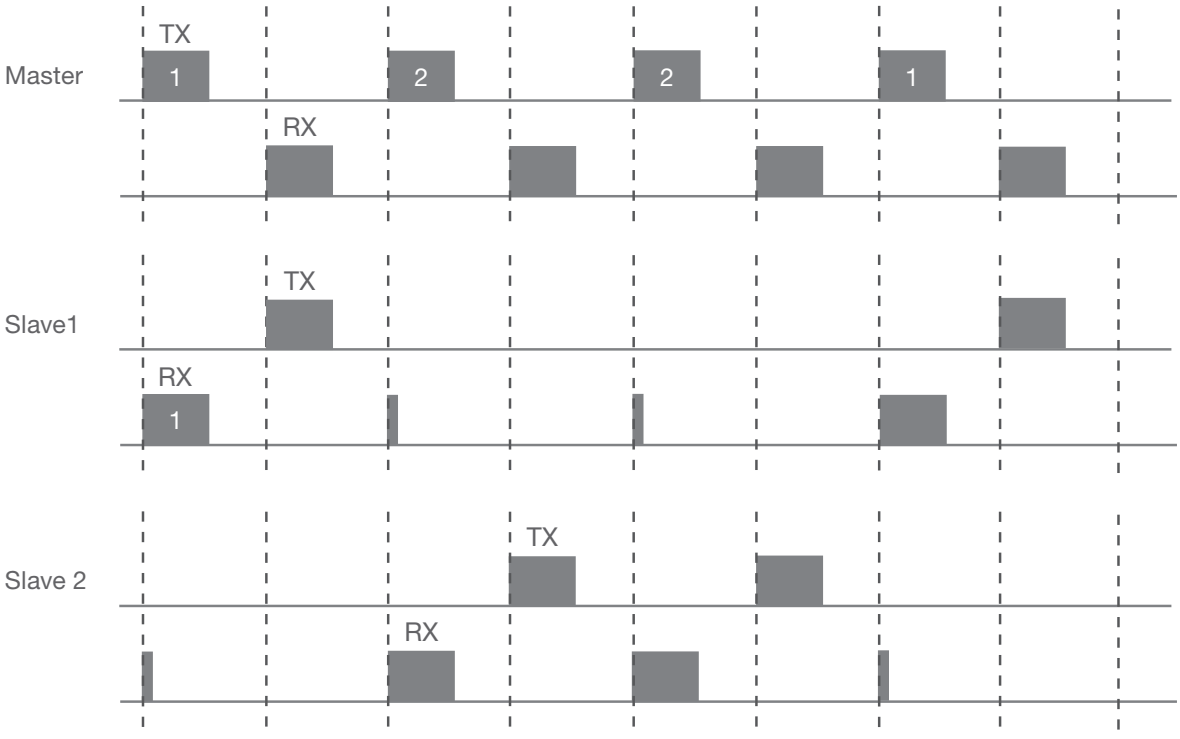

Abb. 5.4 RX/TX timing in multi-slave Konfiguration

TEST

Der Master fragt seine Slaves normalerweise in einem bestimmten Slotabstand ab, dieser wird in der Variable TPOLL gespeichert. TPOLL ist eine globale Intervallangabe, welche für verschiedene Aktionen genutzt und im Link Manager gesetzt wird (u. a. wird hiermit auch die Zeit (Anzahl Slots) begrenzt, die der Master einem Slave hintereinander für Transfers zur Verfügung stellen kann). Jeder Chiphersteller kann diesen Wert selbst definieren, für die hier verwendeten Module liegt dieser bei maximal 25ms (entspricht 40 Slots) [CSR], dass heißt also, dass der Master seinen Slave ca. alle 25ms erst nach Daten abfragt. [BT4] Dies stellt den Worstcase dar, der zwar nicht immer erreicht wird, die zusätzliche Verzögerung beträgt immerhin trotzdem durchschnittlich zwischen 10ms und 15ms. Die maximale Gesamtlatenzzeit ließe sich damit durch eine Formel darstellen:

TMax = TPoll + TWindowslatenz

Den Datenaustausch stellt das folgendes Szenario dar: Das Mastermodul unterliegt keinen Beschränkungen im Netz, es entscheidet selbst, wann es senden darf und wann nicht. Ein Datenpaket kann also ohne Rücksicht auf die anderen Teilnehmer sofort im nächsten Sendeslot an den zu adressierenden Slave gesendet werden. Da dieser immer auf Pakete vom Master horcht nimmt er diese auch sofort entgegen. In der Gegenrichtung sieht dies dann anders aus, der Slave muss auf eine Sendeaufforderung (Pollpaket) vom Master warten und darf erst dann mit dem Sendevorgang beginnen. Es kann also der Fall eintreten, wo der Slave ein Paket senden will, kurz nachdem er dem Master signalisiert hat es lägen keine Daten vor. Nun muss der Slave die rund 25ms abwarten um die Daten loszuwerden, dies erhöht folglich die Latenzzeit der Übertragung.

Die zweite Messung wurde mit den gleichen Parametern wie die vorangegangene mit ebenfalls gleichen Umgebungsvoraussetzungen durchgeführt. Lediglich die Kabellänge zwischen dem PC und den Geräten wurde vergrößert um den Abstand zwischen den Modulen und damit die Funkstrecke zu vergrößern. Nach den Spezifikationen für eine RS232-Verbindung beträgt die erlaubte Maximallänge für serielle Verbindungen bei einer Geschwindigkeit von 115 kbps 1,5 Meter. Es wurden hier jedoch keinerlei Beeinträchtigungen festgestellt, Übertragungsfehler durch das lange Kabel gab es augenscheinlich nicht. Die Entfernung zwischen den Modulen betrug somit ca. 8 Meter bei entsprechender Kabellänge. Die Durchführung entsprach ebenfalls der bereits oben genutzten Weise.

TEST

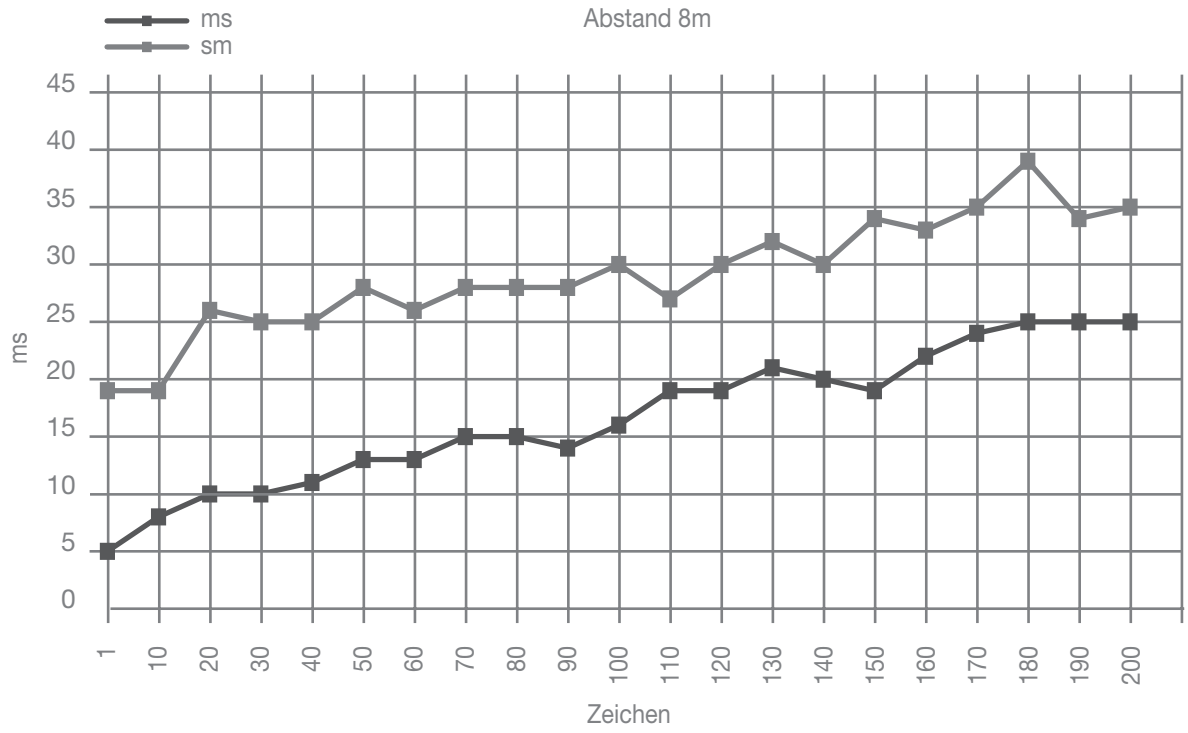

Abb. 5.5 Latenzzeit in beiden Richtungen, Abstand ca. 8 Meter

Im Vergleich zu der ersten Messung fallen kaum Unterschiede auf, die Kurven verhalten sich ähnlich, die Steigung ist insgesamt gleich.

VERGLEICH UND EVALUIERUNG

6.1 Messung mit einem Logikanalysator

In diesem Kapitel wird ein Vergleich zwischen den durchgeführten Tests aus dem vorherigen Kapitel mit der Software und Tests mit herkömmlicher Messhardware durchgeführt. Mittels dieser „per Hand“ durchgeführten Tests soll zudem die Korrektheit und Funktionalität der Software hinsichtlich der Anforderungen aus der Analyse geprüft werden.

Für die „per Hand Messung“ wird zusätzliche Hardware benötigt. Um möglichst genaue Daten zu erhalten wird ein kompakter Logikanalysator, der GoLogic von NCI (www.nci-usa.com), genutzt. Dieser kann theoretisch auf bis zu 32 Kanälen parallel maximal 1 Million Pegelwechsel aufzeichnen und speichern. Die TTL-Pegel (Transistor-Transistor-Logik), welche an den RS232-Treibern (hier ist der Hardwarebaustein gemeint, dieser heißt im Allgemeinen auch Treiber, nicht zu verwechseln mit einem Softwaretreiber) der Entwicklerboards anliegen, sind von den Boards über die I/O-Pins abgreifbar. Dies stellt die genaueste Messmethode dar, da die Kabelstrecke vom PC zu den Modulen und auch alle Verzögerungen des Betriebssystems außen vor gelassen werden. Für die Latenzzeitbestimmung kann auf der grafischen Oberfläche der zugehörigen Software der Abstand zwischen den Daten auf der TX Leitung des Senders und der RX Leitung des Empfängers gemessen werden. Die nachfolgende Grafik veranschaulicht dieses Vorgehen:

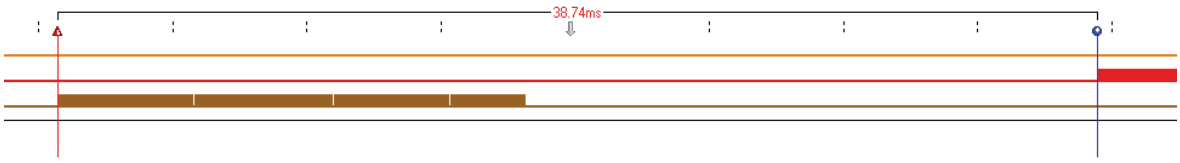

Abb. 6.1 Latenzmessung im GoLogic, Genauigkeit: 1/100 ms; braun: TX, rot: RX

Auf Grund der wesentlich höheren Genauigkeit dieser Methode wurde für die Ermittlung der Durchschnittsverzögerungszeit pro Messdurchgang nur 10 anstatt 15 Einzelmessungen durchgeführt. Der restliche Ablauf blieb erhalten, ebenso die Datenmenge und Art. Da das GoLogic nicht über die Möglichkeit verfügt, die Kabel zu verlängern und somit die Messstrecke von 8 Metern nicht realisierbar ist, wurde lediglich die Testreihe mit kurzem Abstand durchgeführt. Ein stark ver-

VERGLEICH UND EVALUIERUNG

ändertes Verhalten zwischen den Messaufstellungen wird auf Grund der vorangegangenen Messungen bei kurzen Distanzen sowieso nicht erwartet.

Das Ergebnis für die Master->Slave Richtung zeigt ein besseres Ergebnis was die Kurvendynamik betrifft. Legt man nun die ermittelten Kurven beider Messmethoden übereinander zeigt sich, dass die Kurven unterschiedlich stark „ausschlagen“, der Kurvenverlauf insgesamt jedoch gleich ist.

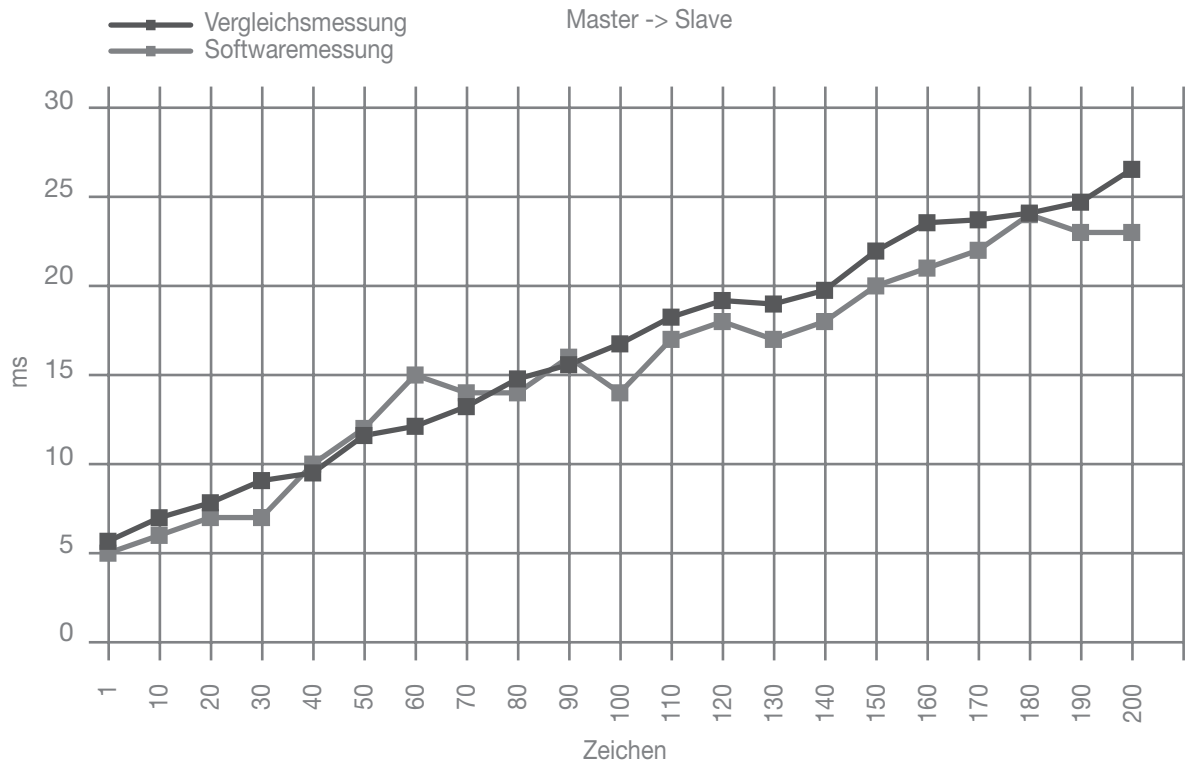

Abb. 6.2 Latenzzeiten im Vergleich: Master -> Slave

Die Abweichung der herkömmlichen Methode von der softwaregestützten Methode liegen bei maximal 1 bis 2ms. Grund hierfür werden schlicht die doch teilweise erheblichen Schwankungen unterworfenen Windowslatenzzeiten sein.

Die Vergleichsmessung in der Slave->Master Richtung zeigt ein ähnliches Bild.

VERGLEICH UND EVALUIERUNG

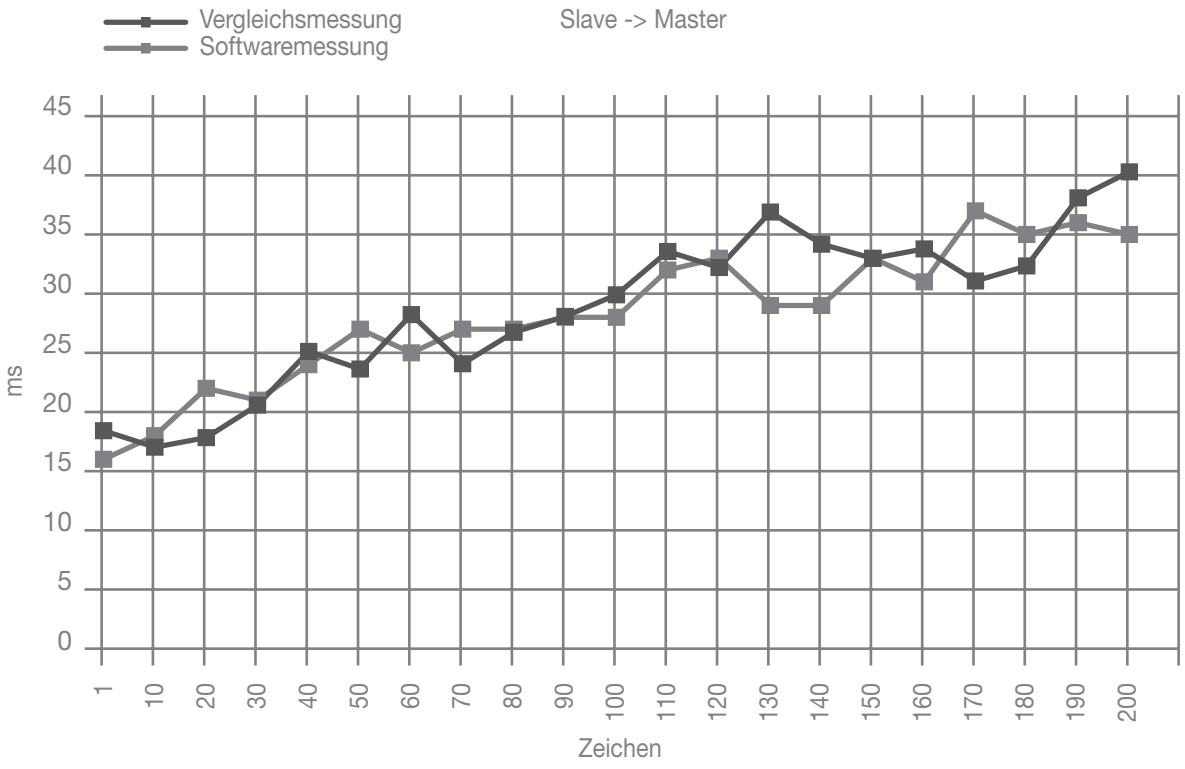

Abb. 6.3 Latenzzeiten im Vergleich: Slave -> Master

Wesentliches Kennzeichen dieser Messung ist, dass die Vergleichsmessung ebenso schwankt wie die softwaregestützte Methode. Die relative Abweichung der Kurven untereinander ist ähnlich wie oben, der schwankende Kurvenverlauf beider Kurven ist auf den in 4.5 dargestellten Pollinterval zurückzuführen.

6.2 Bewertung der Software

Anhand der Grafiken lassen sich die Messergebnisse beider Methoden gut vergleichen. In ihrer Aussage sind die Kurvenverläufe alle relativ gleich, die absoluten Werte weichen nur geringfügig voneinander ab. Dies ist tolerierbar, zumal der Aufwand und die damit verbundene Zeitersparnis bei der Softwaremethode gegenüber der Handarbeit weitaus geringer ist. Zum Vergleich: Für die Ermittelung der Messdaten unter Zuhilfenahme der erstellten Mess-Software wurden ca. 30 bis 40 Minuten benötigt. Die andere Methode verbrauchte hier ein vielfaches, nämlich ca. 2,5 Stunden, wobei hier die Anforderung an die Konzentrationsfähigkeit des Anwenders zudem höher liegt.

VERGLEICH UND EVALUIERUNG

Die gemittelte Abweichung zwischen den Kurven der entsprechenden Methode beträgt in der Regel 1 bis 2ms, in der Slave-to-Master Richtung ist diese stellenweise höher. Zurückzuführen ist dies aber nicht nur auf die nicht konstante Verzögerungszeit des Windowssystems, sondern auch auf die steten Schwankungen unterliegende drahtlose Verbindung. Teilweise wird der Effekt dadurch noch verstärkt, in der Grafik 5.3 ist dies an der Position (140/29) gut zu erkennen. Speziell in der Slave-to-Master Richtung fallen die Schwankungen in beiden Kurven relativ stark aus, hier kommt neben allen vorherigen Ungenauigkeiten noch der Sachverhalt des zyklischen Pollings hinzu (siehe Kapitel 4.5).

Gut zu erkennen ist allerdings, dass es keine Rolle spielt, ob hier die Messungen mit einer Genauigkeit im µs-Bereich oder im ms-Bereich liegen. Die Software vermag auch mit gerundeten Werten respektable Ergebnisse im Vergleich zur aufwendigeren Methode zu erzielen. Letztendlich ist noch festzustellen, dass beide Messmethoden lediglich eine Momentaufnahme liefern können, das genaue Latenzzeitverhalten kann nicht vorhergesagt werden. Je nach Verbindungsqualität, „Luftverschmutzung“ (hohe Konzentration mehrerer Geräte) durch andere Drahtlosfunktechniken oder sonstigen Beeinflussungen kann sich die Latenzzeit jederzeit verändern. Bereits während der Aufnahme der Messdaten mittels des GoLogic kamen teilweise Latenzzeiten von 18 bis 23ms bei einer Blocklänge von 150 Zeichen vor (Master-to-Slave), wobei der Mittelwert schließlich bei ca. 21ms lag. Eine wirklich störungsfreie Umgebung ist heutzutage aber nur noch in einem abgeschirmten Labor darstellbar, dies würde jedoch den Zweck der Software verfehlen.

5.3 Nutzbarkeit mit anderen Herstellern

Neben der Firma Stollmann E+V GmbH gibt es natürlich einige weitere Firmen, die in der Erzeugung von Bluetoothendgerätelösungen auf dem Markt vertreten sind. Die Mess-Software wurde zwar nur für die Geräte von Stollmann angefertigt, jedoch sollte sie prinzipiell auch für alle weiteren Geräte mit serieller RS232-Schnittstelle nutzbar sein. Sinn dieses Kapitels ist es zu zeigen, dass die Software dieser Anforderung gerecht werden kann.

VERGLEICH UND EVALUIERUNG

Als zusätzliche Hardware werden zwei Geräte der Firma Phoenix Contact verwendet. Phoenix ist ein deutschstämmiger Hersteller für Datenübertragungsterminals speziell für industrielle Anlagen, Roboter, etc. Die Geräte verfügen über je eine seriell Schnittstelle nach RS232/RS485 und ein Bluetoothmodul der Klasse 1 bzw. 2. Ein interner Speicher sichert die Geräteadressen bekannter und als sicher kategorisierter Bluetoothgeräte, die Devices können sich dadurch ohne gesonderte Befehle mit anderen Geräten automatisch verbinden, dies macht die Option 6 der Software somit für diesen Fall überflüssig. Versetzt man die Module in den Konfigurationsmodus, können diese jedoch auch über einige herkömmliche, sowie produktspezifische AT-Befehle angesprochen und konfiguriert werden, dies stellt aber nicht den erdachten Regelbetrieb dar.

Abb. 6.4 Bluetooth-Konverter, Phoenix Contact

Der restliche Verlauf ist unverändert zu dem bereits bekannten geblieben, es gibt eine, mit der Software erstellte Messreihe und eine durch das GoLogic ermittelte. Das System wird zunächst wieder angelernt und dann können direkt Daten über die Funkstrecke gesendet werden. Die Module nutzen hierbei ebenfalls das reguläre SPP-Protokoll.

Da die Bluetoothgeräte von Phoenix für einen Verbindungsaufbau auf automatischen Aufbau und automatisches Akzeptieren eingestellt werden müssen, ist es nicht ohne weiteres möglich herauszufinden, welches Gerät die Master- und welches die Slave-Rolle übernimmt. Weiterführende Tests haben jedoch ergeben,

VERGLEICH UND EVALUIERUNG

dass dieser Sachverhalt keine Rolle spielt, die Latenzzeiten sind, im Gegensatz zu den BlueEva+Boards von Stollmann E+V GmbH, in beiden Richtungen gleich. Warum das so ist kann nur gemutmaßt werden. Da Phoenix einen anderen Bluetoothchip verwendet (NXP, ehemals Philips), kann bereits hier der Unterschied konstruktionsbedingt sein. Weiterhin kann dies auch eine Designentscheidung der Softwareprotokolle seitens von Phoenix sein. Genaueres kann leider nicht ermittelt werden, Phoenix legt verständlicherweise seine Enwicklungsdokumentation nicht offen dar. Aus diesem Grund wird hier nur eine Messreihe in einer Richtung ausgewiesen, fernerhin soll diese Messreihe nur die Kompatibilität der Software mit Geräten anderer Hersteller belegen.

Für den Messeinsatz konnte die Software bereits ihren ersten Vorteil ausspielen, die Industrieprodukte von Phoenix konnten natürlich keine geeigneten Konnektoren vorweisen, über welche man leicht die Signale für die Aufzeichnung durch das GoLogic abgreifen könnte. Somit mussten die Gehäuse geöffnet, zerlegt und die Messpunkte direkt an den Lötstellen am seriellen Stecker angebracht werden. Dies stellte sich Anfangs als sehr fehleranfällig heraus, da die hierfür verwendeten Klammern leicht abrutschten und somit der Kontakt verloren ging. Diese Fehlerquelle ist bei Einsatz der Softwaremessung nicht vorhanden. Die Geräte wiesen, wie es meist bei Komponenten für die Steuerung industrieller Anlagen der Fall ist, bereits geeignete Hardwareschnittstellen vor.

VERGLEICH UND EVALUIERUNG

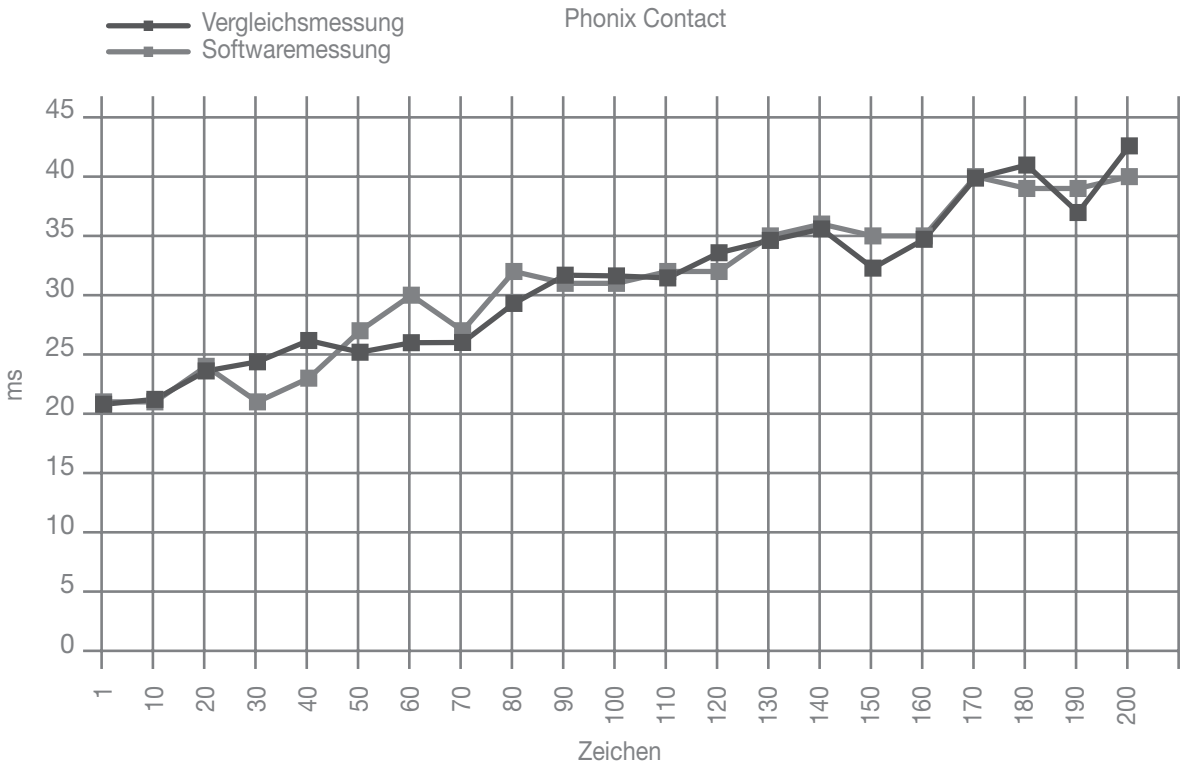

Abb. 6.5 Latenzzeiten im Vergleich: Phoenixmodule

Das Resultat ist durchaus positiv, die verschiedenen Messungen zeigen ein sehr ähnliches Bild vom Kurvenverlauf, wie es bereits mit den Entwicklerboards von Stollmann zu sehen war. Zwar entstehen auch hier Abweichungen zwischen den Messmethoden untereinander, jedoch sind diese wiederum begrenzt. Die softwaregestützte Lösung hat also nicht nur ein weiteres Mal ihre Korrektheit gegenüber der unter Einsatz des Logikanalysators ermittelten Werte bewiesen, sondern auch, dass der Einsatz nicht auf eine bestimmte, kleine Produktgruppe festgelegt zu sein scheint.

FAZIT

7.1 Zusammenfassung

Das Grundthema der Arbeit war die Erstellung eines Mess-Systems für Bluetooth SPP-Verbindungen. Am Anfang stand die Erläuterung zur Motivation das System zu erstellen und die Deklaration von Fragestellungen, die den positiven Abschluss der Entwicklung verhindern könnten. So war es zu Beginn nicht abzusehen, welche Einflüsse das zu verwendende Windowsbetriebssystem auf das Gesamtverhalten haben würde.

Kapitel 2 führte in die für das Verständnis benötigten Grundlagen ein. Hier wurde ein umfassender Einblick auf die internen Ablaufprozeduren und Aufbau von einem auf Windows NT basierenden Betriebssystem aufgezeigt. Mögliche Probleme wurden hier bereits sichtbar, zumal der Hardwarezugriff für jegliche Anwendung einer bestimmten Ablaufprozedur unterworfen wird, welche zeitlich vom Entwickler nicht vorhersehbar ist. Weiterhin wurde ein grober Einblick in die Drahtlostechnologie Bluetooth geschaffen und näher auf die für die zu messende Verbindungsart, das Serial Port Profile, eingegangen. Das Kapitel schloss mit einer Vorstellung der benötigten Funktionen aus der WindowsAPI ab; die hier dargestellten Hilfsfunktionen ermöglichen es einem Entwickler auf zum Beispiel jegliche Computerhardwareschnittstellen zuzugreifen, was ihm ja sonst vom System verwehrt wird.

Kapitel 3 definierte zunächst nochmals den Zweck der zu erzeugenden Software und erläuterte dann die funktionalen, sowie nichtfunktionalen Anforderungen an das System. Zudem zeigte es die durch die schwankenden Latenzzeiten entstehenden Grenzen bezüglich des Sinnvollen. So wäre es beispielweise nicht nützlich gewesen, die Messgenauigkeit auf µs aufzulösen.

Das nächste **Kapitel (4)** befasste sich mit dem Design der für das Mess-System benötigten Software. Der Gesamtaufbau sowie die einzelnen erzeugten Methoden wurden detailliert vorgestellt. Am Ende wurden einige Tests in der später einzusetzenden Umgebung durchgeführt. Die Auswertung des erwarteten und des unerwarteten Verhaltens fand hier ebenso statt, sowie auch die Erklärung des Phänomens, dass die Latenzzeit vom Master zum Slave konstant geringer ist als die in der Gegenrichtung.

FAZIT

Kapitel 5 befasste sich mit einem Vergleich zwischen der erstellten Software und einer weiteren, herkömmlichen Messmethode. Dieser Vergleich stellt zudem auch die Evaluierung dar, ob das erzeugte System die in Kapitel 3 geforderten Lösungen umgesetzt hat und ob die Grundaufgabe, die Erstellung eines Mess-Systems für Bluetooth SPP-Verbindungen, erfüllt wurde. Ebenso wurde getestet, ob das System auch mit anderen Geräten konkurrierender Hersteller nutzbar ist.

7.2 Fazit

Das Ziel, ein System für die Messung von Latenzzeiten in Bluetooth SPP-Verbindungen auf einem Windowssystem kann im Hinblick auf die erzielten Ergebnisse als erfolgreich gewertet werden. Entgegen den zu Beginn aufkommenden Befürchtungen, das Betriebssystem Windows könnte die Ergebnisse soweit verfälschen, dass die Ergebnisse unbrauchbar wären, bestätigten sich glücklicherweise nicht. Es exstieren zwar Beeinflussungen der Laufzeiten durch Windows, allerdings waren diese im Durchschnitt stabil, so dass diese für die Messresultate herausgerechnet werden konnten. Für die Messgenauigkeit bedeutet dies jedoch, dass die Auflösung nur im Bereich von Millisekunden sinnvoll ist.

Im Bezug auf die ermittelten Messdaten stellte sich heraus, dass die erstellte Software nur geringe Unterschiede zu den mit herkömmlichen Mitteln gesammelten Daten aufwiesen. Die Abweichungen der einzelnen Messungen untereinander sind zwar immer vorhanden, jedoch zeigt sich, dass die durchschnittlichen Werte in beiden Methoden zueinander relativ gleich sind. Bereits mit wenigen Messwerten (hier 15) vermag die Software ähnliche Ergebnisse zu erzielen, wie die präzisere Messung mittels Logikanalysator.

7.3 Ausblick

Natürlich kann die erstellte Software nicht als fertig im Sinne des Bedienkomforts (nichtfunktionale Anforderungen) angesehen werden. Es stellt sich nun die Frage, ob nicht doch eine Windows-GUI einsetzbar ist, welche die Bedienung verbessern könnte. Hier könnte auch die Auswertung der Messergebnis grafisch

FAZIT

erfolgen, denkbar wäre hier bereits eine Übertragung in eine Exceltabelle oder proprietäre Tabelle/Diagramm.

Weiterhin sinnvoll könnte eine Testautomatisierung sein, wo der gesamte Messdurchgang einem Algorithmus folgend durchgeführt werden kann. Ein Zusatz an Komfort wäre zudem ein automatisches Auslesen der Bluetoothadressen des angeschlossenen Slaves. Alle Geräte verfügen in Ihrem AT-Befehlssatz über ein Kommando, mit welchem die eigene Bluetoothadresse und der Gerätename zurückgegeben werden kann. Mittels des Befehls „ATI“ geben alle Stollmannprodukte ihre eigene Gerätebezeichnung wieder als Textstring zurück. Leider ist der Auslesebefehl für die Bluetoothadresse nicht bei allen Produkten gleich, somit müsste erst mittels eines Stringvergleiches das Produkt identifiziert und dann mit dem richtigen AT-Befehl die Bluetoothadresse ausgelesen werden. Für die hier eingesetzten Module wäre dies „AT**BOAD“, die Rückgabe ist wiederum ein Textstring. Andere Hersteller haben sich aber nicht zwangsweise an diese Befehle gehalten. Obwohl die Anweisung „ATI“ laut AT-Befehlsliste im Anhang eigentlich standardmäßig genutzt werden sollte, weicht beispielsweise Phoenix Contact hier von dem Schema ab und nutzt eigens erstellte Befehlsversionen. Die Abfrage des Gerätenamens ist hier nämlich „AT*AGLN?“. Dies bedeutet also für die Implementierung dieses Features, dass für den Komfort eine erweiterbare Herstellerdatenbank erzeugt werden müsste, wo der Anwender zunächst den Hersteller des zu testenden Devices auswählen müsste.

Eine weitere sinnvolle Modifikation wäre es, das System auf einer eigenständigen Hardwareplattform zu realisieren. Die in dieser Arbeit verwendeten Entwicklerboards würden sich für eine Portierung auf den dort platzierten Atmelprozessor eignen. Dieser verfügt bereits über genügend Leistungsreserven um zwei Hardwaretimer und zwei USARTs, welche direkt an die Schnittstellentreiber angeschlossen sind, zu bedienen. Somit könnte das Device als Ersatz für einen PC genutzt werden. Da hier auch der Zugriff auf die Hardware direkt erfolgt und die Abfolge auch nicht durch einen vorgefertigten Scheduler reglementiert wird, kann die verfälschende Latenzzeit im System selbst sehr weit vermindert werden. Selbst wenn weiterhin noch eine solche Latenzzeit auftaucht, so wird diese immer gleich sein, was dann auch Messgenauigkeiten in den Mikrosekundenbereich ermöglichen könnte – ob diese Genauigkeit sinnvoll wäre lässt sich jedoch

FAZIT

anhand der bereits erfahrenen Ergebnisse bezüglich der Schwankungen in der Bluetoothfunkstrecke bezweifeln. Allerdings wäre dann der Ansatz, ein System zu erzeugen, welches auf einem Laptop einsetzbar ist, nicht mehr gegeben.

Die letzte zu erörternde Erweiterung ist sicherlich die Möglichkeit, die Distanz zwischen den Geräten zu vergrößern um längere Funkstrecken erzeugen zu können. Wie bereits erwähnt ist die RS232-Schnittstelle bezüglich der Kabellänge bei 115 kbps auf lediglich 1,5 Meter festgelegt. Dies kann durch eine Weiterentwicklung der hier genutzten Schnittstelle umgangen werden. In der Industrie hat sich der sogenannte „Profibus“ nach RS485 in der Maschinensteuerung durchgesetzt. Über diese Schnittstelle kann eine maximale Entfernung von 1200 Metern überbrückt werden, wobei die Übertragungsgeschwindigkeit maximal 10Mbit betragen kann. Die Geschwindigkeit nimmt jedoch bereits nach 15 Metern proportional ab, sodass sich folgendes Szenario darstellt: Wenn man davon ausgeht, dass der PC gemäß dem oben beschriebenen Messaufbau mittig zwischen den Modulen (Class 1) platziert ist, ergibt dies bei maximaler Funkdistanz von 100 Metern eine Kabellänge von jeweils 50 Meter. Folglich ist eine maximale Geschwindigkeit von ca. 2,5 Mbit möglich [RS485]. Zum Vergleich: RS232 würde bei gleicher Länge lediglich eine Baudrate von 9.600 bps bieten. Allerdings ist die RS485-Schnittstelle nicht im gleichen Maße verbreitet beziehungsweise vorhanden wie RS232, es ist jedoch möglich mit Hardwarekonvertern von RS232 auf RS485 zu arbeiten. Hierbei ist allerdings mit hoher Wahrscheinlichkeit davon auszugehen, dass davon die Latenzzeiten beeinflusst werden und somit diese Alternative nur bedingt realisierbar ist.

FAZIT

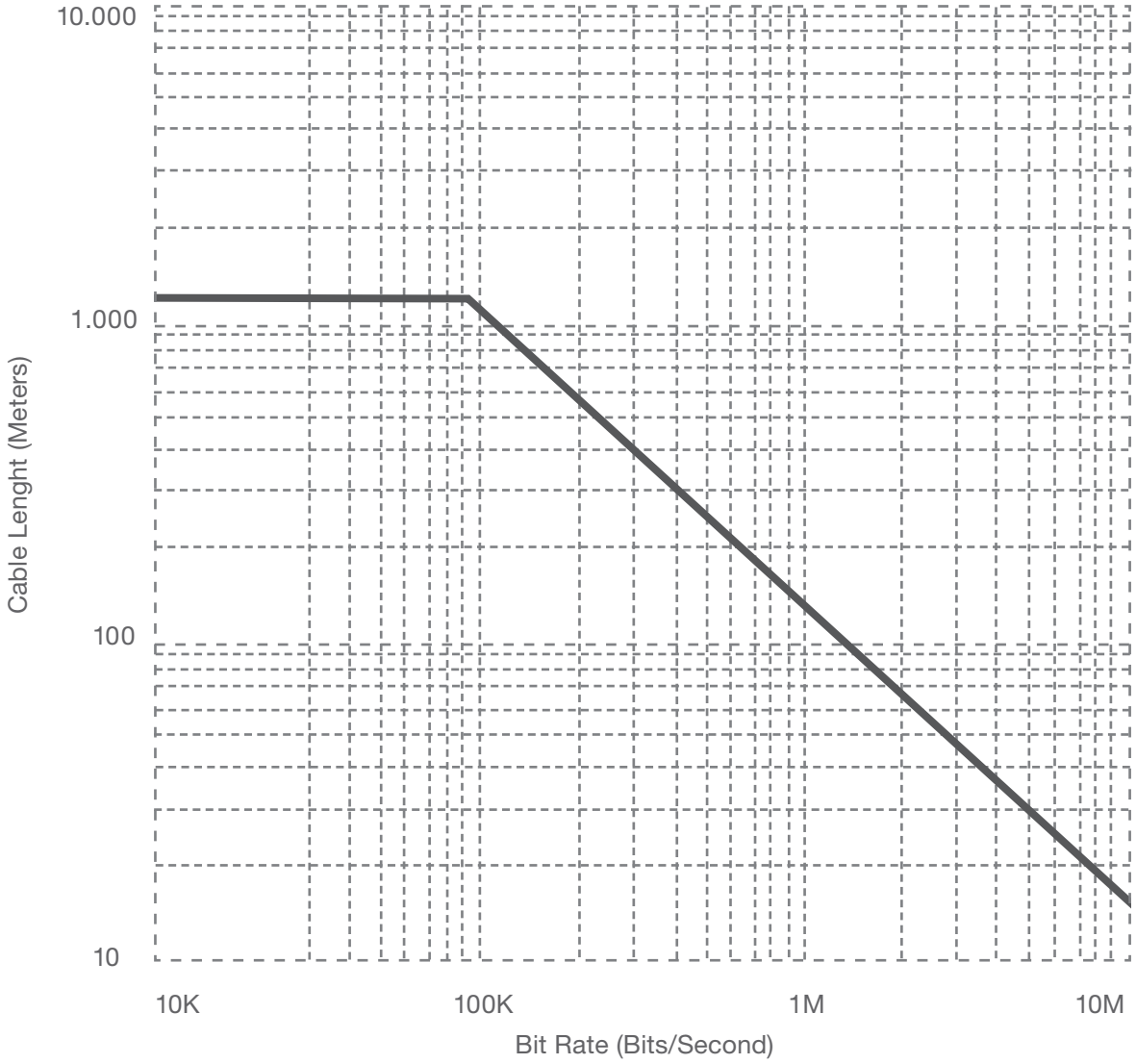

Abb. 7.1 Geschwindigkeit von RS485 bezüglich Kabellänge

GLOSSAR

Layer = Schicht

Deadlocks = Stillstand des Systems durch z.B. eine Endlosschleife.

EIA/TIA-232-E = Norm der Electronic Industry Association/Telecommunications Industry Association zur Nutzung des seriellen Datenports, ehemals RS-232 (RS=recommended standard).

ETSI Standard TS 07.10= Besser bekannt als TS 101 369 v6.3.0(GSM 07.10 v6.3.0 1997), hier wurden Standards für das GSM Netz festgelegt; da Bluetooth besonders im drahtlosen Telefoniebereich überzeugen sollte, wollte man schlicht Problemen aus dem Weg gehen und die vorhandenen Standards im gleichen Zug umsetzen, obwohl die Techniken wenig gemein haben.

UUID = Universally Unique Identifier – eindeutiger Identifizierer, bestehend aus einer 16Byte Zahl, unterteilt in fünf Gruppen.

OBEX = (OBject EXchange); Kommunikationsprotokoll und dient der Datenübertragung zwischen zwei Geräten. OBEX arbeitet im klassischen Client-Server-Modell. Es wird heute überwiegend bei serieller Kommunikation (Bluetooth, Infrarot) zu mobilen Endgeräten (PDA, Smartphone) eingesetzt. Nach OSI-Modell gehört es zur Sitzungsschicht (Schicht 5) und ist somit transparent zu der tatsächlichen Kommunikationsmethode.

UNICODE 8 = Festgelegter Zeichensatz, ähnlich wie ASCII.

Bluetoothstack = Das grundlegende Betriebssystem für ein Bluetoothmodul.

Handle = (engl.) „Henkel“, „Griff“, ein numerischer Identifikationswert vom Typ long.

LSB = Least Significant Bit

MSB = Most Significant Bit.

NT (Windows) = NT wurde nach Veröffentlichung auf “New Technology” umgetauft, dies hat aber nur Marketinggründe. Verbreitet ist auch „Network Technology“. Die wahre Bedeutung ist aber „N-Ten“, hierbei handelt es sich um einen Emulator für Windows. Seinerzeit wurde von Intel der i860 (RISC) Prozessor

GLOSSAR

entwickelt (Codename N10), da dieser aber noch nicht verfügbar war musste man sich eines Emulators bedienen. Windows NT ist also ein Betriebssystem, welches ursprünglich für eine ganz andere Plattform entwickelt wurde und nun einen Emulator beinhaltete [N10].

USART= Universal Synchronous Asynchronous Receiver Transmitter, erzeugt den für die RS232-Schnittstelle benötigten Datenrahmen.

QUELLENVERZEICHNIS

9.1 Quellenverzeichnis

[ABB] ABB Stotz Kompakt, http://www.industrialnetworks.ch/dateien/aussteller/abb-ch/2CDC142001B0102.pdf, Letzter Zugriff: 26.6.

[AGV] http://www.megatech.at/ireds-27445.html, Letzter Zugriff: 25.6.

[AT] http://www.itwissen.info/definition/lexikon/Hayes-Befehlssatz-Hayes-command-set.html, Letzter Zugriff: 16.6.

[AW] http://www.informationsarchiv.net/statisch/wlan/geschichte.html, Letzter Zugriff: 17.6.

[BT1] Das Bluetoothhandbuch, Jörg F. Wollert, 2002, Franzis'

[BT2] Bluetooth 1.1 – Connect without cables SE, Jennifer Bray & Charless F Sturman, 2002, Prentice Hall PTR

[BT3] http://german.bluetooth.com/Bluetooth/Technology/, Letzter Zugriff: 28.6.

[BT4] Specification of the Bluetoothsystem Version 2.0 + EDR

[BT5] Bluetooth aktuell – Technik und Anwendungen, Prof. Dr. Jörg Wollert, Elektronik 23/2001

[CSR] CSR's Implementation of HCI on BlueCore, CSR 2001

[GOLEM] http://www.golem.de/0505/38239.html, Letzter Zugriff: 09.06.

[N10] http://wapedia.mobi/de/Windows_NT, Letzter Zugriff: 9.7.

[ROCLA] http://www.rocla.com, Letzter Zugriff: 10.6.

[RS232] Serial Port Complete, Jan Axelson, 2007, Lakeview Research LLC

[RS485] Serial Port Complete, Jan Axelson, 2007, Lakeview Research LLC, S. 88, Abb. 6-4

[SENA] http://www.wirelessdevicesolutions.com/solutions/industrialautomation/case_studies/#wireless_cnc_dnc_network, Letzter Zugriff: 25.6.

QUELLENVERZEICHNIS

[SCHED] http://book.itzero.com/read/microsoft/0507/microsoft.press.microsoft.windows.internals.fourth.edition.dec.2004.internal.fixed.ebook-ddu_html/0735619174/ch06lev1sec5.html, Letzter Zugriff: 21.7.

[WIKI] http://de.wikipedia.org/, Letzter Zugriff: 20.7.

[WINAPI] http://msdn.microsoft.com/en-us/library/aa363201(VS.85).aspx, Letzter Zugriff: 18.6.

[WNT] Inside Windows NT, Helen Custer, 1993, Microsoft Press

9.2 Bilderquellenverzeichnis

Kapitel 2

(2.1) [WNT], S. 17
(2.2) [WNT], S. 18
(2.3) [WNT], S. 19
(2.4) [SCHED]
(2.5) [WNT], S. 26
(2.6) [BT2] S. 9
(2.7) [BT2] S. 35, [WIKI]
(2.8) http://nedt.at/docs/bluetooth/fdd.jpg, Letzter Zugriff: 25.6.
(2.9) http://www.rooijen.de/studium/praxis/eins/image002.jpg, Letzter Zugriff: 13.6.
(2.11) http://www.hochschule-bochum.de/fileadmin/media/fb_e/labore/srlab/uploads/Artikel/200123s050.pdf, Letzter Zugriff: 13.6.
(2.12) http://german.bluetooth.com/NR/rdonlyres/21815114-AFD1-4BB3-81F6-FFD3F4AF8490/986/SPP_SPEC_V12.pdf, Letzter Zugriff: 13.6.
(2.13) [BT1] S. 240 Tab. 56
(2.14) Evaluationboard BlueEva+C11/G2, Stollmann E+V GmbH, 2008
(2.16) http://de.wikipedia.org/wiki/EIA-232, Letzter Zugriff: 11.6.

Kapitel 5

(5.4) Specification of the Bluetoothsystem Version 2.0 + EDR, S. 335, Abb. 8.6

QUELLENVERZEICHNIS

Kapitel 6

(6.1) Screenshot aus der GoLogic Software

(6.4) Abbild PSI-Bluetooth-Konverter, PSI-WL-RS232-RS485/BT, Phoenix Contact

QUELLENVERZEICHNIS

9.3 AT-Befehlssatz

AT	Attention	Achtung
A	Answer Immeditate	Sofortantwort
A/	Repeat Last Command	Letztes Kommando wiederholen
A:	Redial	Wahlwiederholung
B	Bell/CCITT-Compatibility	Bell/CCITT-Kompatibilität
D	Dial Number	Rufnummer wählen
E	Echo Characters	Kommando-Echo
F	Communications Format	Übertragungsformat
G	Talk	Gespräch
H	Hook Switch Control	Verbindung auflösen
I	Inquiry	Identifizierung
J	Reserved	Reservieren
K	Call Timer	Verbindungszeit
L	Loudness	Lautstärke
M	Speaker Control	Lautsprecherkontrolle
N	Dial Stored Number	Registerwahl
O	Online	Online
Q	Quit Mode	Ruhe
S	Set Register	Register einstellen
V	Result Codes	Rückmeldungen
X	Extended Result Codes	Erweiterte Rückmeldung
Y	Long Space Disconnect	Auslösen mit Break
Z	Recall User Configuration	Konfigurationsprofil laden
&A	Automatic Rate Detection	Automatische Geschwindigkeits-anpassung
&B	Delayed Make Busy	Verzögertes „Busy-Signal“

QUELLENVERZEICHNIS

&C	Carrier Detect Control	DCD-Optionen
&D	DTR-Options	DTR-Optionen
&E	Error Correction	Fehlerkorrektur
&F	Factory Configuration Load	Werksgrundeinstellung
&G	Guard Tone Selection	Guard-Ton-Wahl
&H	Help Command	Hilfe anfordern
&I	Constant Speed Interface	Konstante Geschwindigkeit
&J	Telephone Jack	Anschlussbelegung
&K	Modem Flow Control	Steuerungsverfahren
&L	Leased Line Connection	Anschluss an Stromwege
&M	Asynchronous/Synchronous	Asynchron/Synchron-Mode Select
&P	Make/Break Dial Pulse Ratio	Pulse/CTS-Optionen
&R	RTS/CTS-Options	RTS/CTS-Optionen
&S	DSR-Options	DSR-Optionen
&T	Test Commands	Test-Kommandos
&W	Write Configuration to	Konfiguration speichern Memory
&X	Synchronous Transmit Clock	Synchron-Taktquelle Source
&Z	Store Telephone Number	Telefonnummer speichern

Abb A.1 AT-Befehlssatz:
http://www.itwissen.info/definition/lexikon/Hayes-Befehlssatz-Hayes-command-set.html

VERSICHERUNG DER SELBSTÄNDIGKEIT

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 05. August 2008

Bent Mucha