Beuth Hochschule für Technik Berlin

Bachelorarbeit

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences

WN Serial COM Port Tester

Erstprüfer:	Prof. Dr. Voß
Zweitprüfer:	Prof. Dr.-Ing. Rozek

Eingereicht am
4. September 2013

Eingereicht von
Matthias Hansert 764369

Inhaltsverzeichnis

Kapitel 1

Einleitung

Gefordert ist ein Programm, das auf Wincor Nixdorf Mainboards und auf Erweiterungskarten(COM-Karten) implementierte RS232/COM-Port-Hardware, die zugehörigen Hardware-Treiber und das BIOS, indirekt testet. Das Testtool wird in der Programmiersprache „C / C++“ entwickelt und wird auf folgenden Betriebssystemen implementiert:

- Windows XP SP3 x86, x64
- Windows 7 x86, x64
- WinPE

Unter den möglichen Fehlerfällen wird das Tool Kabel- und Stecker-Fehler(Wackelkontakt durch schlechten mechanischen Kontakt der Stecker und der Buchsen) erkennen. Elektrische Fehler, die sich bei der Übertragung über ein langes Nullmodemkabel in Form von Paritätsfehlern zeigen oder die durch Windows-interne Ressourcenprobleme(Shared Interrupt) hervorgerufen werden, werden erkannt.

Das Programm soll die Schwächen anderer Serial-Ports-Tools ergänzen, wie zum Beispiel die eingeschränkte Skriptfähigkeit und die Testautomatisierung. Diese aktuell verwendeten Tools unterstützen keine Zähler, keine Uhrzeit mit aktuellem Datum als Sendedaten und keinen Zufallsgenerator. Zudem können sie nicht von einen schreibgeschütztem Medium gestartet werden. Das Tool soll direkt über die Windows API die COM Ports testen, da die Kunden von der Firma Wincor Nixdorf und die Applikationen die COM Ports über diese API steuern. Ein weiterer Vorteil der Nutzung der Windows API ist, dass die Scripting- und Automatisierungsoptionen besser implementierbar sind. Durch die Voraussetzung der Nutzung der Windows API, ist die Installation oder Implementierung des Tools auf einem Linux-System nicht möglich.

Das Programm besteht aus einer GUI, in der alle Programm- und Schnittstellenparameter (Baudrate, Stoppbits, Paritätsbit, Datenbits, Hardware und Software) eingestellt werden können. Wenn das Programm über die Kommandozeile oder ein Skript gestartet wird, muss eine Konfigurationsdatei oder die Konfigurationsparameter übergeben werden. Wie diese Datei strukturiert sein muss wird im Kapitel 7 behandelt.

Kapitel 2

Aufgabenstellung

Das Tool soll die RS 232 Schnittstelle testen und die Schwächen der vorhandenen Lösung ergänzen. Wincor Nixdorf besitzt schon ein Tool was die Druckertestumgebung testet, dies ist aber Ruby basierend. Das Tool soll mit Funktionsaufrufe der WinAPI programmiert werden, genauso wie die unter die Systeme laufende Anwendung es tun. Die vorhandenen Lösung, wie zum Beispiel ÄGGsoft Serial Port Tester"haben eingeschränkte Einstellungen und sind nicht ausführlich genug für die Voraussetzungen der Qualitätssicherung.

2.1 Module

Es hat folgende funktionelle Einheiten:

- Windows-GUI(Graphical User Interface) mit allen Einstellmöglichkeiten und Start-Button.
- Kommandozeilen User Interface mit passenden Einstellmöglichkeiten, wie auch per GUI und Rückgabewerte (Exitcodes).
- Funktionskern, der aus beiden UI angesprochen wird und die eigentlichen Tests durchführt.

2.2 Laufzeit

Das Programm kann von einem schreibgeschütztem Medium (z.B. Windows-PE-CD) laufen, d.h. alle Schreibfunktionen (Log-Datei) sind per Option abschaltbar. Die Testparameter können in einer Konfigurationsdatei gespeichert und wieder geladen werden. Alle Ereignisse / Meldungen können in einer Log-Datei eingetragen werden. Das Programm startet und testet die Ports automatisch (Skriptmodus).

2.3 Hardware

Folgende Hardware-Kombinationen werden unterstützt (die Reihenfolge hier gibt die Priorität der Implementierung wieder):

1. Test einer Schnittstelle mit Kurzschlussstecker auf einem System

2. Test zweier Schnittstelle mit Null-MODEM-Kabel (RxD/TxD gekreuzt; RTS/CTS gekreuzt, etc.) auf einem System
3. Test zweier Schnittstelle mit Null-MODEM-Kabel auf unterschiedlichen Systemen! Siehe dazu auch „Synchronisierungsprotokoll“...

2.4 Betriebssystem

Das Tool ist entwickelt für und zu testen mit folgenden Zielbetriebssystemen:

- Windows XP SP3 x86, x64; Windows 7 x86, x64; WinPE 2.x (WinXP) / 3.x (Win7)
- Windows 8 nur ein Test, der aussagt ob es kompatibel ist oder noch etwas zu tun wäre

Linux ist hier nicht gefordert.

2.5 Logbuch

Alle Ereignisse / Testschritte können zu Dokumentations- und Analysezwecken in einer Log-Datei gespeichert werden. Auch Fehler, die der Parser in der Test-Datei(.ini) findet, werden in der Log-Datei eingetragen. Das Format ist hier CSV, d.h. es kann direkt in Excel geöffnet werden. Damit sind dann auch Performance-Messungen / -vergleiche zwischen Port auf unterschiedlichen Mainboards einfacher möglich. Der Name der Log-Datei ist für jede Instanz und jedes Testsystem eindeutig zu wählen, zum Beispiel:

<Programmname>_<Computername>_<Master/Slave>_<Port>.csv

Jeder Eintrag bekommt einen Uhrzeit/Datums-Stempel. Eine Zeile sieht dann wie folgt aus:

<Datum>; <Uhrzeit>;<Ereignis>;<Wert>;<Kommentar>

Die ersten beiden Einträge der Log-Datei:

<Datum>; <Uhrzeit>;START;RS232-Port-Nummer; INI-Dateiname als Kommentar

2.6 Programminstanzen

Das Programm kann in mehrfachen Programm-Instanzen gestartet werden, in der Regel läuft bei umfangreichen Tests für jeden zu testenden RS232-Port eine Programm-Instanz. Das vermeidet aufwändige Thread-Programmierung eines einzigen Programms für alle Ports. Das ist keine feste Vorgabe, sollte sich Thread-Programmierung als einfacher erweisen, dann sollte es auch benutzt werden.

2.7 Master / Slave

Jede Programminstanz wird als Master, Slave oder als beides gestartet.

2.7.1 Master-Funktion

Im Master-Modus schickt die Schnittstelle die vorgegebenen Texte / Dateien mit den eingestellten Parametern an den Slave. Der Master wertet die dem Test zugeordnete INI-Datei aus, der Slave folgt nur dem Master. Der Master wartet auf das Echo oder eine Fehlermeldung und bewertet den Testschritt sobald der Test komplett und fehlerfrei empfangen als erfolgreich, sonst als fehlerhaft. In beiden Fällen werden Fehler, Warnungen und Erfolgsmeldungen geloggt.

2.7.2 Slave-Funktion

Im Slave-Modus (Programminstanz auf gleichem System oder zweiten BEETLE/PC) empfängt die Daten komplett, wertet eventuell Fehler aus und schickt entweder die Daten als Echo oder eine Fehlermeldung wieder zurück.

2.7.3 Master / Slave Kombination mit Kurzschlussstecker

Im kombinierten Master / Slave-Modus (Kurzschlussstecker) übernimmt eine Instanz des Programms beides, das heißt, sie schickt sich selber Daten und antwortet ebenso. Programm intern laufen aber die beiden Module (Master / Slave) weitestgehend getrennt.

2.7.4 Master / Slave Ablauf mit „Wobbeln“-Funktion

Die Master-Instanz übernimmt die Steuerung des Ablaufs, der Slave antwortet dem Master. In diesem Testmodus werden durch den Benutzer eingegebenen Baudraten(Unter und Obergrenze) gewobbelt. Der Wobbeln-Ablauf kann über die GUI oder in der INI-Datei beschrieben werden. Der automatische Test erfolgt dann nur für die dort eingetragenen Parameter, Z.B.:

BAUD = 9600 - 115200

Hier werden die Baudrate von 9600 bis 115200 für den Test berücksichtigt, aber 2400, 300, und so weiter nicht. Wenn mehrere Parameter flexibel gesetzt / programmiert sind, dann werden alle Kombinationen getestet. Das kann dann durchaus eine nicht unerhebliche Testzeit beanspruchen.

Die Programminstanz öffnet die zugehörige Schnittstelle automatisch mit der ersten von allen der programmierten Optionen (Baudrate, Parität, etc.) und startet dann einen Testdurchlauf mit diesen Einstellungen. Wenn alle Daten fehlerfrei gesendet und danach auch das Echo fehlerfrei empfangen wurden, dann wird der Port geschlossen, mit der nächsten Kombination geöffnet und wieder Daten gesendet / empfangen. Wenn Fehler (Parität, Offline, etc.) vom Master erkannt oder vom Slave gesendet wurden, dann wird der letzte Schritt wiederholt. Wenn der aktuelle Test schon die x-te Fehler-Wiederholung war, dann stoppt das Programm und zeigt den Fehler an. Dieser Fehler-Zustand darf durch das Programm nicht verändert werden. Der Tester / Entwickler muss der Fehler analysieren.

2.7.5 Synchronisierungsprotokoll

Für den Wobbelmode ist ein (minimales) Protokoll zur Synchronisation nötig. Wenn eine Programminstanz als Master gestartet wird, erhält sie die nötigen Schnittstellenparameter per GUI

oder aus der Konfigurationsdatei. Sie sendet diese in From eines Konfigurationspakets dem Slave. Solche Konfigurationspakete werden immer mit den Standardschnittstellenparametern (96,o,8,1) gesendet. Der Slave synchronisiert sich dann, d.h. programmiert -per WIN-API- seinen zu testenden Port mit den empfangenen Parametern und wartet auf Nutzdaten.

Wenn der erste Test erfolgreich war (Master hat gesendet und der Slave hat ein Echo geschickt), dann schickt der Master an den Slave die Parameter des nächsten Testschritts. Das ist so nötig, weil im Fehlerfall (Echo fehlerhaft empfangen) der Slave nicht mehr synchron wäre, er weiß nur ob seine Daten fehlerfrei gesendet wurden, aber nicht ob der Empfang richtig angekommen ist. Der Slave kann so einfach nur mit Standard-Parametern gestartet werden und folgt dann dem Master.

2.8 Teststruktur

Gefordert ist eine einfache Trennung von Testablauf / Testoptionen und Testobjekt(RS232-Port). Die Teststeuerung wird daher sowohl im Programm selber, als auch über die Kommandozeile implementiert. Der Testablauf für jeden Port selber ist in der zugehörigen Konfigurationsdatei vollständig, aber unabhängig vom zu testenden Port (RS232-/V.24-Schnittstelle), beschrieben. Es können mehrere Konfigurationsdateien auf der Kommandozeile übergeben werden. Die Schnittstellenummer wird über die Kommandozeile festgelegt. So kann der gleiche Testablauf (INI-Datei) für jeden Port einzeln / parallel durchgeführt werden.

2.9 Benutzerschnittstelle

2.9.1 GUI

In der GUI sind alle Programm- und Schnittstellenparameter einstellbar und können jederzeit aus der GUI in eine Konfigurationsdatei gespeichert werden. Der Windows-Fenster bietet also implizit auch die Funktion eines Konfigurationseditors mit eingebauter Syntaxüberprüfung.

2.9.2 Kommandozeile

Wenn die Kommandozeile keine Parameter bekommt, wird automatisch die GUI dargestellt. Die Kommandozeile nimmt folgende Optionen entgegen:

- -f INI_Datei
- -f INI_Datei /Port

In der INI Datei ist der Testvorgang beschrieben. Wenn keine „/Port“ angegeben wurde, dann wird der Port in der INI Datei getestet. So sieht ein Programmaufruf aus:

- SerialPortTester.exe KOMPLETT.INI
- SerialPortTester.exe KOMPLETT.INI /COM3

2.10 Skriptfähigkeit

Per Kommandozeile gestartet, ist das Programm BATCH-/Script-fähig, d.h. es startet ohne weitere Benutzereingaben, loggt alle Ereignisse und auch Hinweise -zur Analyse nach dem Test- in die zugehörige Datei und beendet sich ebenso selbstständig. Fehler, Warnungen und Hinweise werden auf der Konsole ausgegeben. Als Option kann es auch im Hintergrund ohne jede Anzeige rennen.

2.11 Programmierung der Testparameter

Die Testparameter können direkt per GUI oder per INI-Datei eingestellt werden. Die aktuelle Testkonfiguration und der Ablauf kann in einer INI-Datei gespeichert und wieder geladen werden. Eine mit GUI erstellte INI-Datei kann über Kommandozeile geladen werden. Diese INI-Datei hat die Windows-übliche Struktur mit „[“ und „]“ und ist per Notepad (oder irgendeinem anderen Editor) zu bearbeiten.
Sie enthält alle Informationen zur Steuerung der Programminstanz, das heißt alle einstellbaren Schnittstellenparameter inklusive Wobbelnoption (von-bis):

Baudrate = 1200-9600; Stopbbits = 1; Datenbits = 8; RTS_CTS=yes; etc.
Baudrate = min-max; Stopbbits = min-max; etc.

Für MIN/MAX siehe Schnittstellenoptionen. Timeout-Werte (für Echo-Antworten auf Datenpakete) sollten keine festen Zeiten programmiert sein, das Programm berechnet die Wartezeiten automatisch aus der aktuellen Baudrate. Verzögerungszeiten zwischen zwei Datenblöcken/Testschritten inklusive Zufallswert ist wünschenswert.

2.11.1 Schnittstellenoptionen zu Test

Folgende Werte sollen für den Benutzer einstellbar sein:

- Baudrate
- Parität
- Datenbits
- Stopbits
- Flusssteuerung

Das Programm kann hier mit Variablen MIN und MAX umgehen. Wenn beispielsweise die Zeile „BAUDRATE = MAX“ oder „BAUDRATE = MIN“ programmiert wurde, dann sucht das Programm die maximale bzw. minimale Baudrate, die angegebene RS232-Port unterstützt und nutzt sie für den Test. Diese Werte werden aus „dem System“ –beispielsweise Registry oder MSPORTS.DLL direkt ausgelesen. Manuell können diese MIN/MAX-Werte über den Gerätemanager ermittelt werden. Wenn keine Parameter übergeben wurden, dann startet das Programm mit diesen Standardparametern: 9600,odd,8,1,RTS/CTS (von der Wincor Nixdorf Anzeige BA66).

Die Wobbelnoption wird in Form einer VON-BIS-Zeile implementiert:

BAUDRATE=300;2400;9600;115200	diese vier Werte werden getestet
BAUDRATE=MIN-MAX	alle möglichen Werte
BAUDRATE=300-9600;115200	alle Werte von 300 bis 9600 und 115200 im Test

2.11.2 Sendedaten aus Datei

Als Testdaten werden feste Texte (in der Regel ASCII Zeichen) oder auch Pattern aus einer Datei gesendet. Die Dateipfade / Sendetexte sind als Testparameter in der GUI oder Test-Datei) einstellbar. Wenn keine Sendedaten übergeben wurde, dann sendet das Programm zufällige, aber auf einem Protokollanalysator lesbare, ASCII-Zeichen: 0x20 ... 0x7F .

2.12 Fehlererkennung und Behandlung

Folgende mögliche Fehlerfälle muss ein solches Tool erkennen:

- Kabel-/Stecker-Fehler („Wackelkontakt“ durch schlechten mechanischen Kontakt Stecker/-Buchse)
- Elektrische Fehler, die sich bei Übertragung über langes Null-MODEM-Kabel im Form von Paritäts-, Rahmenfehlern oder ähnliches zeigen
- Dauerhaft gezogene Kabel / Kurzschlussstecker
- Windows-interne Ressourcenprobleme (Shared Interrupts)

Bei Erkennung eines Übertragungsfehlers wird dieser in die Logdatei eingetragen und / oder auf der Konsole angezeigt und danach der letzte Block wiederholt. Nach der x-ten erfolglosen Wiederholung (Wert aus Test-Datei) erfolgt Testabbruch

Kapitel 3

Fachlichesumfeld

3.1 RS 232 (Radio Sector 232)

Die Quellen dieses Kapitel sind aus „http://de.wikipedia.org/wiki/RS-232; 25.08.2013“ und „Joe Campbell: V 24 / RS-232 Kommunikation. 4. Auflage. Sybex-Verlag GmbH“.

Der RS 232 ist ein Standard der in den 60er Jahren von die US-amerikanische Standardisierungskomitee Electronic Industries Association(EIA) bearbeitet und definiert wurde. Bei diesem Standard haltet es sich um eine serielle Schnittstelle die für die serielle Kommunikation zwischen Rechnern und Modems in einer Punkt-zu-Punkt Verbindung(über Telefonleitungen) diente.

3.1.1 Definition

Unter der RS 232 Standard ist die Verbindung zwischen eine Datenendeinrichtung(DEE, zum Beispiel das Terminal) und eine Datenübertragungseinrichtung(DÜE, zum Beispiel einem Modem) und dessen Parameter definiert. Unter Parameter sind der Spannungspegel, Übertragungsprotokoll(Handshake), Stecker und Timing zu verstehen.

Die Übertragung ist Bit-seriell, das heißt, Bits werden hintereinander verschickt in einer Datenleitung. Eine bestimmte Menge an Bits entspricht ein Wort. Ein Wort entsteht aus ein Startbit, die tatsächliche Datenbits(Nutzdaten), ein oder zwei Stopbits und die Parität. Die Parität ist in den Standard nicht definiert, aber wird benutzt um Fehler zu erkennen und beheben. Die Parität wird hier betrachtet, da sie für diese Arbeit relevant ist und wird später genauer erklärt.

3.1.2 Übertragung

Der Startbit meldet den Empfänger das die Übertragung anfängt. Start- und Stopbits habe inversen Pegeln. Also ist eine Leitung in Ruhezustand oder hat die Stopbits erhalten, wird durch die Ankunft einer inversen Signalflanke aufmerksam gemacht, dass danach Nutzdaten ankommen werden. Bei der Ankuft eines Startbits tastet der Empfänger die Nutzdaten mit seiner Bitrate(Bits pro Sekunde). Die Nutzdaten sind die Bitdarstellung von einem ASCII Zeichnen. Die Nutzdaten sind einstellbar, die Übertragung kann jeweils von 4 bis 8 Bits pro Zeichen eingestellt werden. Danach kommt mindestens ein Stopbit. Es können auch 1,5 Bits und zwei Bits eingestellt werden. 1,5 Bits kling sehr ungewöhnlich, aber damit ist gemeint, dass die Mindestdauer der Pause zwischen zwei ankommende Wörter 1,5 Bitzellen entspricht.

Um genauer die Übertragung von einem Zeichen zu verstehen, folgender Beispiel. Wird eine Charackter 'z' übertragen, mit einem Startbit, acht Datenbits und zwei Stopbits, ist die gesamt Länge der Übertragung elf Bits lang. 'z' entspricht laut ASCII Kodierung den Wert 122 dezimal und „7A“ hexadezimal(0x7A). Die Bitdarstellung für 0x7 lautet „0111“ und für 0xA „1010“, zusammen für 0x7A = „01111010“. Die Übertragung im diesem Standard folgt mit dem LSB(Less significant Bit) zuerst, also werden die Bits vertauschst zu „0101 1110“. Davor wird eine Null zugefügt als Startbit und am Ende zwei Einsen als Stopbits. Das Ergibt, dass für die Übertragung eines 'z' die Bitreihenfolge „0 0101 1110 11“ verschickt wird.

Die Datenübertragung unter RS 232 ist asynchron, das bedeutet es existiert kein gemeinsamer Takt. Die Bitraten zwischen Sender und Empfänger dürfen um wenig Prozent von einander abweichen, sonst wird der Empfänger das Wort zu schnell / langsam abtasten und falsch interpretieren. Dagegen muss die Baudrate bei Sender und Empfänger genau gleich sein. Beide Begriffe sind nicht zu verwechseln. Eine Bitrate definiert die übertragene Bits pro Sekunde und die Baudrate die übertragene Symbole pro Sekunde, wo jedes Symbol als definierte messbare Signaländerung im physischen Übertragungsmedium definiert ist. Da die Übertragung im diesem Standard binär ist, ist ein Symbol als ein Bit definiert. Dieses hat als Folge, dass im diesem Spezialfall die Bitrate und die Baudrate gleich sind.

Die Baudrate wird vom Benutzer frei wählbar eingestellt. Dafür muss der Benutzer die Kabellänge betrachten. Weil es sich um eine Spannungsübertragung handelt, muss der Leistungswiederstand und die Kapazität des Kabels beachtet werden. Je länger das Kabel, desto starker nimmt die Spannung ab. Nach Erfahrungswerte von Texas Instruments ist bei einer Baudrate von 9600, eine maximale Kabellänge von 152 Meter möglich. Bei 115200 muss das Kabel kürzer als 2 Meter sein.

Der Empfänger muss die Datenübertragung anhalten können, wenn er keine Daten mehr verarbeiten kann. Dieser Handshake wird softwaretechnisch oder über die Hardware mit Steuerleitungen.

Bei der Softwarelösung werden am Sender spezielle Steuerzeichen gesendet. Dieses Protokoll ist als „Xon/Xoff“ bekannt. Es ist nur möglich dieses Protokoll zu benutzen wenn die Steuerzeichen(Xon = 0x11 und Xoff = 0x13) nicht in der Nutzdaten vorkommen.

Bei der Hardwarelösung signalisieren Sender und Empfänger sich gegenseitig ihren Status. Solche Protokoll besteht zum Beispiel aus fünf Steuerleitungen(TxD, RxD, GND, RTS und CTS).

Damit triviale Fehler vermieden werden, müssen Sender und Empfänger die gleichen Einstellungen haben. Das heißt dass die Baudrate, Stopbits, Parität, Handshake und Kabellänge müssen stimmen.

3.1.3 UART

Universal Asynchronous Receiver Transmitter(UART) ist eine elektronische Schaltung zur Realisierung von digitalen seriellen Schnittstellen(für diese Arbeit der sogenannte COM Port). Der

UART ist zum senden und empfangen von Daten über eine Datenleitung vorgesehen. Im industriellen Bereich ist der UART unter die RS 232 Standard sehr verbreitet. Auch in Wincor Nixdorf wird diese Schnittstelle zum steuern der COM Ports benutzt.

3.1.4 Paritätsbit

Wie schon erwähnt, ist die Parität nicht im RS 232 Standard definiert, aber sie ist für diese Arbeit relevant. Das Paritätsbit dient zur Erkennung fehlerhafte Übertragungen. Die Parität kann gerade(even) oder ungerade(odd) sein und wird durch die Anzahl an Einsen in einer Bitfolge bestimmt. Ist eine gerade Parität festgelegt, wird bei einer gerade Anzahl an Einsen eine Null angehängt, bei ungerade Anzahl eine Eins. Genau das Gegenteil geschieht bei ungerade definierte Parität. Nach unseren Beispiel mit 0x7A wird bei gerade Parität eine Eins („0 0101 1110 **1** 11“) angehängt.

3.1.5 Leitungen und Stecker

Am Anfang wurde der 25-polige D-Sub-Stecker verwendet. Viele dieser 25 Leitungen sind reine Drucker und Terminal-Steuerleitungen aus der elektromechanischen Zeiten, somit waren sie für die modernere Peripherie überflüssig. So hat sich der 9-polige D-Sub-Stecker(COM Port) etabliert. Dieser Stecker war nicht ursprünglich für diesen Standard gedacht, sondern wurde von IBM, als Notlösung in einen anderen Standard, um Platz zu sparen entwickelt. Der Stecker ist daher unter EIA/TIA-574 zu finden. Für die EIA-232-Datenübertragung werden selten andere Stecker benutzt und in Wincor Nixdorf ist das auch keine Ausnahme.

Der 9-polige D-Sub-Stecker besteht aus folgenden Leitungen:

TxD, TX, TD	: Transmit Data, Leitung für ausgehende Daten
RxD, RX, RD	: Recieve Data, Leitung für ankommende Daten
RTS	: Request To Send, Sendanforderung
CTS	: Clear To Send, Sendeerlaubnis
DSR	: Data Set Ready, Einsatzbereitschaft
GND	: Ground, Signalmasse
DCD, CD, RLSD	: Data Carrier Detect, Erkennung einlaufende Daten
DTR	: Data Terminal Ready, Datenendeinrichtung ist bereit
RI	: Ring Indicator, Datenverbindungsaufbau

EIA-232 ist eine Spannungsschnittstelle, also werden die logische Null und Eins durch positive / negative Spannungen vertreten. Die Datenleitungen (TxD und RxD) benutzt eine negative Logik. Spannung zwischen -3V und -15V repräsentieren eine logische Eins. Signale zwischen -3V und +3V gelten als nicht definierte Signale. Die logische Null wird als eine Spannung zwischen +3V und +15V interpretiert.

Bei den Steuerleitungen, bezogen auf dem Empfänger, wird ein Signal zwischen +3V und +15V als aktiv betrachtet, und inaktiv zwischen -3V und -15V. Beim Sender üblicherweise ±12V.

Um Sender und Empfänger zu verbinden, gibt es verschiedene Kabelvarianten, abhängig von wer Sender und wer Empfänger ist. Verbindet man ein Rechner (in der Regel mit einem Stecker) zu einem Modem (mit einer Buchse) ist ein 1:1 Kabel nötig. Sind zwei Rechner mit einander Verbunden, so ist die Rede von einem Nullmodem Kabel, wo die Leitungen gekreuzt sind. Durch

ein Loopback-Stecker bzw. Kurzschlussstecker wird die Sendeleitung direkt and die Empfangsleitung der gleichen Schnittstelle umgeleitet. So ein Stecker wird für die Entwicklung und Test von Kommunikationsanwendungen und Hardware(UART) benutzt.

3.1.6 RS 232 in Wincor Nixdorf

Die BEETLE Systeme haben alle?????? ein integrierten UART im Chipsatz. Diese COM Ports werden „Onboard Ports“ benannt. Je nach System sind zwei bis sechs COM Ports eingebaut. Wenn ein Kunde weitere Schnittstellen benötigt, kann über den PCI Bus die Anzahl an Ports erweitert werden. Diese Erweiterung findet statt durch eine Sunix PCI Karte mit vier oder acht COM Ports.
Intel SOL
ITE
stromversorgung ba66 kassenlade

3.2 Microsoft Windows API

Die Quellen des folgenden Kapitel sind aus „Programming Windows, Charles Petzold, 1995, 5. Auflage“ und „Visual C++ 2010, Dirk Louis, 2010“.

3.2.1 Definition

Die Microsoft Windows „Application Programming Interface“(Schnittstelle zur Anwendungsprogrammierung) ist ein Programmteil, dass vom Windows Betriebssystem den Benutzern und vor allem Entwicklern angeboten wird, um Programme an das Betriebssystem anbinden zu können. Ein Betriebssystem (Microsoft Windows, Mac OSX, Linux, unter anderem) ist für Entwickler und Programmierer durch die API definiert. Somit kann eine Applikation über die API alle Funktionsaufrufe machen, die ein Betriebssystem anbietet. Nicht nur Funktionen sind in einer API definiert, sondern bestimmte Datenstrukturen und Datentypen durch das Kommando *typedef* wie *LRESULT* oder *CALLBACK*.

Mit fast jedes neue Microsoft Betriebssystem wird die Windows API erweitert und abgeändert. Die erste API, bekannt als *Win16*, für die 16-Bit Versionen von Microsoft Windows. Für Windows 1.0 hatte die API etwa 450 Funktionsaufrufe. Bei der Zeit von Windows 98 wurde die API auf 32-Bit und mehrere tausende Funktionsaufrufe. Ab Windows XP „x64 Edition“ und Windows Server 2003 wurde die API auch auf 64-Bit erweitert.

Der hauptsächliche Unterschied zwischen die 16, 32 und 64 Bit Versionen von der API entstand durch die verschiedene Speicher und Prozessor Architekturen. Unter der 16-Bit Architektur war die Registergröße 16 Bit unter die bekannte Prozessoren von Intel 8086 und 8088. In der 32-Bit Architektur, 32 Bit bzw. in der 64-Bit, 62 Bit groß. Die Windows API ist in der Programmiersprache „C“ geschrieben. Deswegen war unter die 16-Bit Architektur der Datentyp *int* „nur“ 16 Bit lang(Zahlen von -32.768 bis 32.767) . In der Speicherverwaltung bestanden Speicheradressen aus einen 16-Bit Segment und einen 16-Bit offset Zeiger. Für Programmierer war diese Verwaltung sehr Umständlich, da der Programmierer genaue Unterscheiden musste, zwischen *long* oder *far* und *short* oder *near* Zeiger.

Ab die 32-Bit Architektur entstand die „Flat Memory Model“, wo der Prozessor direkt die gesamte Speicheradressen ansprechen konnte, ohne Speichersegmentierung oder Pagingschemas. Somit wurde auch der *int* Datentyp auf 32 Bitgröße(Zahlen von -2.147.483.649 bis 2.147.483.647) definiert. Programme geschrieben unter eine 32-Bit Architektur benutzen einfache Zeigerwerte um direkt die Speicheradresse ansprechen zu können. Bei der Umstellung von 16-Bit auf 32-Bit blieben viele Funktionsaufrufe gleich, aber manche brauchten eine Umstellung auf 32-Bit. Wie zum Beispiel das graphische Koordinatensystem für GUI Darstellungen.

Aus Kompatibilitätsgründe sind die API’s Rückwärts kompatibel. Die Kompatibilität entsteht durch eine Übersetzungsschicht. Es gibt zwei Wege der Übersetzung. In den ersten Weg, werden 16-Bit Funktionsaufrufe durch eine Übersetzungsschicht in 32-Bit Funktionsaufrufe umgewandelt und dann vom Betriebssystem bearbeitet. Der andere Weg führt genau in der anderen Richtung.Die 32-Bit Funktionsaufrufe durch die Übersetzungsschicht und wandelt diese in 16-Bit Funktionsaufrufe, und werden dann vom Betriebssystem bearbeitet.

Die Benutzung der API ist nicht die einzige Möglichkeit Anwendung für die Windowsbetriebssysteme zu programmieren. Aber durch die Benutzung der API ist eine bessere Leistung, mehr Macht und Flexibilität in das Ausnutzen der Betriebssystemfunktionen garantiert. Durch die Verwendung der API versteht man Windows als Betriebssystem besser. Man kann Anwendungen auch in Visual Basic oder Borland Delphi schreiben, wo die objektorientierte Grundlagen von Pascal den Programmierer viel arbeitet erleichtern kann. Aber das Stapeln von Programmierschichten über die API versteckt nur die Komplexität der API, und früher oder später wird man im Programm mit dieser Komplexität konfrontiert.

API gegenüber .NET Framework

Microsoft hat dieses Framework speziell für die Windows-Plattformen entwickelt. Es ist eine Virtuelle Maschine als Laufzeitumgebung für Microsoft Windows Anwendungen. Dieses Framework gleich in vieles die Java Virtual Machine. Das .NET Framework besteht aus eine Laufzeitumgebung und die .NET Framework-Bibliothek. Aus Sicht des Anwenders hat sich nichts geändert, aber für die Programmierer vieles. Das .NET Framework ist auf C++ und C# basierend, und im Gegensatz zur API, objektorientiert. Die Framework-Bibliothek besteht aus verschiedenen Klassenbibliotheken wie die Windowa Forms, Windows Presentation Foundations (GUI), Webdienste, unter anderem. Ein großer Vorteil ist die Portierung der Programme, dafür muss das .NET Framework installiert sein. Das ist für dieses Projekt essentiell, denn es soll auf Schreibgeschütze Medien ausführbar sein (Win PE) und aus diesem Grund für meine Lösung nicht betrachtet.

3.2.2 Windows.h

Die Window.h Headerdatei ist die Masterdatei, dass alle andere Headerdateien inkludiert. In diesen Dateien sind die Funktionsaufrufe, Konstanten, Typdefinitionen und Datenstrukturen für das Windowsbetriebssystem definiert. Diese Headerdateien sind die Teil der Dokumentation für den Programmierer. Wer die Headerdateien kennt, weiss auch was man unter windows programmieren kann. Die wichtigsten Headerdateien sind:

- WINBASE.H Kernenfunktionen
- WINDEF.H Typdefinitionen
- WINNT.H Typdefinitionen mit Unicode Unterstützung
- WINUSER.H Funktionen für die Benutzerschnittstelle
- WINGDI.H Graphische Schnittstelle

3.2.3 WinMain

C/C++ Programme fangen mit eine *int main()* Funktion an. Da die API auf *C* implementiert ist, fangen Anwendungen mit Windows als Zielsystem mit eine ähnliche Funktion an. Unter Windows ist die *main* Funktion unter „WINBASE.H“ als:

```
int WINAPI WinMain(
        HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        LPSTR lpCmdLine,
        int nShowCmd
        );
```

Als erstes fällt auf der Bezeichner *WINAPI*. Dieser Datentyp ist definiert als *WINAPI __stdcall*. *__stdcall* ist eine Aufrufkonvention um Funktionen aus der Win32 API aufzurufen. Die *WinMain* Funktion bekommt als Parameter zwei *HINSTANCE* Variablen, ein *LPSTR* und ein *int*.

HINSTANCE ist ein Handle auf eine Instanz. Das ist die Basisadresse von einem Modul im Systemspeicher(eine 32-Bitzahl, dass auf ein Objekt zeigt). Das erste Parameter ist dann eine Instanz auf die aktuelle Anwendung. Das zweiter Parameter wurde nur unter Win16 benutzt und ist unter die Win32 API irrelevant, es wird automatisch auf *NULL* gesetzt. Das dritte Parameter enthält die Befehlszeilenargumente als Zeichenfolge. Aus dieser Zeichenfolge wertet das Programm aus, wie es ausgeführt werden soll. Das vierte und letzte Parameter ist ein Flag und gibt an wie das Anwendungsfenster angezeigt werden soll (Minimiert, Maximiert oder Normalgröße).

3.2.4 Graphic User Interface (GUI)

Durch die grenzwertige Hardware(Speicher und Prozessoren) waren alle Betriebssysteme Kommandozeileorientiert. Dies sollte sich aber ändern durch die Recherche von Xerox Palo Alto Research Center(PARC). Mitte der 70 Jahren wurde in Xerox PARC nach graphische Benutzeroberflächen recherchiert. Aus diesen Ergebnisse profitierten Macintosh und Windows, und bauten darauf Ihre Betriebssysteme mit graphischen Benutzeroberflächen(Mac OS und Windows). Heutzutage kann man sich als Benutzer kaum eine Computerwelt ohne Benutzeroberflächen vorstellen. Die Windows API hat seit der Ankündigung(1983) und Veröffentlichung(1985) von Windows als Betriebssystem Funktionsaufrufe in die Headerdateien für die Programmierung von GUI's. Die Benutzung der API für die Programmierung von GUI's ist vielleicht ältmodisch", aber immerhin sehr genau und präzise. Im Gegensatz zu „GUI Builders“ wird kein unnötiger, und oft für Programmierer, unverständlicher Code geschrieben. Der Überblick und Verständnis der GUI Elemente wird durch die direkte Benutzung der API vereinfacht. Vor allem wird das Verständnis wie eigentlich eine GUI unter Windows und Windows als Betriebssystem funktioniert, durch die Verwendung der API deutlicher.

3.2.5 Aufbau einer GUI

Nachdem die *WinMain* Funktion die nötige Parameter bekommt und diese vom Programm ausgewertet werden, muss die GUI als solche gebaut und registriert werden. Eine GUI besteht aus drei Teile.

Initialisierung und Erzeugung der GUI

Damit der Benutzer ein Programmfenster sehen kann, muss dieses zuerst deklariert und initialisiert werden. Danach wird dieses Fenster an das System bekannt gegeben in dem man es registriert. Um ein Fenster zu deklarieren muss man eine Variable der Struktur *WNDCLASS* erzeugen. Diese Struktur definiert in *WINUSER.H* beinhaltet verschiedene Variablen die die Eigenschaften des jeweiliges Fenster beschreiben.

```
typedef struct
{
                UINT            style ;
                WNDPROC         lpfnWndProc ;
                int             cbClsExtra ;
                int             cbWndExtra ;
                HINSTANCE       hInstance ;
                HICON           hIcon ;
                HCURSOR         hCursor ;
                HBRUSH          hbrBackground ;
                LPCTSTR         lpszMenuName ;
                LPCTSTR         lpszClassName ;
}WNDCLASS,    * PWNDCLASS ;
```

Die zwei wichtigsten Variablen dieser Struktur sind der zweite und letzte. Die zweite Variable *WNDPROC lpfnWndProc* beschreibt die Fensterprozedur. Genaueres zu dieser Variable wir demnächst erklärt. Die letzte Variable *LPCSTR lpszClassName*, beschreibt die Klasse des Fensters. Die Variable *HINSTANCE hInstance* muss den Leser bekannt sein. Diese Variable wird auf den Wert gesetzt, welches das Programm über die *WinMain* bekommen hat. Alle andere Variablen der Struktur beschreiben wie das Fenster aussehen soll und sind für das Verständnis weiterhin irrelevant.

Nachdem die *WNDCLASS* Struktur deklariert und initialisiert wird, muss diese registriert werden. Durch den Aufruf der Funktion *RegisterClass*, die als Übergabe Parameter ein Zeiger auf eine *WNDCLASS* Struktur hat, wird dem System bekannt gegeben, dass ein Fenster aufgebaut werden soll (mit dem gesetzten Eigenschaften der *WNDCLASS* Struktur). Wenn das Registrieren erfolgreich war, muss das Fenster noch erzeugt werden. Das System kennt das Fenster, aber es ist noch nicht sichtbar.

Damit ein Fenster sichtbar wird, muss der Programmierer die *CreateWindow*[1] Funktion aufrufen.

```
HWND WINAPI CreateWindow (
                                In_opt  LPCTSTR lpClassName,
                                In_opt  LPCTSTR lpWindowName,
                                In      DWORD dwStyle,
                                In      int x,
                                In      int y,
                                In      int nWidth,
                                In      int nHeight,
                                In_opt  HWND hWndParent,
                                In_opt  HMENU hMenu,
                                In_opt  HINSTANCE hInstance,
                                In_opt  LPVOID lpParam
);
```

Diese Funktion ist sehr wichtig bei der Erzeugung von graphischen Oberflächen, denn man kann alle Arten von Fenstern damit erzeugen. Damit wird gemeint, dass diese Funktion nicht nur

[1] http://msdn.microsoft.com/en-us/library/windows/desktop/ms632679(v=vs.85).aspx; 27.08.2013

„Hauptfenster“, sondern auch die einzelnen GUI Elemente erzeugt. Das erste Parameter beschreibt dieses Verhalten. Die *CreateWindow* Funktion erwartet eine null terminierte Zeichenkette. Diese Zeichenkette ist immer eine vordefinierte Systemklasse. Zum Beispiel, um ein Knopf zu erstellen, muss hier als Parameter „button“ angegeben werden. Wenn der Programmierer ein von ihm erzeugtes Fenster darstellen will, muss er deswegen vorher im System eine Fensterklasse registrieren. Aus diesem Grund muss der Programmierer eine Fensterstruktur *WNDCLASS* registrieren. Um danach ein selbst gebautes Fenster zu erzeugen, muss der Programmieren das registrierte Fensterklassenname(*WNDCLASS lpszClassName*) angeben (anstatt „button“ oder andere vordefinierte Fensterklassen).

Der zweite Parameter beschreibt der dargestellter Name des Fenster. *DWORD dwStyle* ist eine Bitmaske die den Darstellungsart des Fensters beschreibt. Die Darstellungsarten sind als konstante Werte (Window Styles[2])in die *WINUSER.H* Headerdatei definiert. Um ein „traditioneles“ Fenster zu erzeugen muss hier *WS_OVERLAPPEDWINDOW* angegeben werden. Die Nächste vier Parameter Beschreibe die Positionierung im Bildschirm und die Breite und Höhe des Fensters.

HWND hWndParent beschreibt ein Handle auf ein anderes Fenster. Im Falle, dass dieses Fenster das Hauptfenster ist, wird hier NULL übergeben, sonst das Handle auf das „Parent Windows“. Das *HMENU hMenu* ist für Hauptfenster irrelevant. Es ist aber wichtig für Unterfenster(Popup Fenster oder GUI Elemente wie Knöpfe). *HMENU* ist eine Typdefinition für ein Handle auf ein Fenstermenü oder ein Unterfenster. Somit kann das Hauptfenster eine Kommando eines GUI Elements erkennen und auswerten, mehr dazu unter „Die Message Loop“. Das *HINSTANCE hInstance* wurde schon vorher erläutert und das letzte Parameter beschreibt extra Information wenn es nötig ist, üblicherweise wird hier NULL angegeben.

Die *CreateWindow* Funktion liefert als Rückgawert ein *HWND*. Dies steht für Handle auf ein Fenster. Somit kann der Programmierer und Programm auf ein gewünschtes Fenster zugreiffen. Um die erwähnte Parameter genauer zu verstehen folgende Beispiele. Hat man ein Hauptfenster, so ruft man die *CreateWindow* Funktion folgendermaßen:

```
HWND hwnd_Fenster = CreateWindow (

szAwendungsName,                    //Fensterklasse vom Programmierer registriert
"Das ist eine Hauptfenster",        //Text auf der Titelleiste
WS_OVERLAPPEDWINDOW,                //Stil des Fensters
CW_USEDEFAULT,                      //Position in der X Koordinate des Bildschirmes
CW_USEDEFAULT,                      //Position in der Y Koordinate des Bildschirmes
CW_USEDEFAULT,                      //Lange des Fensters
CW_USEDEFAULT,                      //Breite des Fensters
NULL,                               //Kein zugehöriges Hauptfenster
NULL,                               //Keine Identifikationsnummer
hInstance,                          //Programminstanz
NULL                                //keine extra Information
);
```

Will der Programmierer ein GUI Element (wie zum Beispiel ein Knopf) im Hauptfenster darstellen, muss die Funktion mit diesen Parametern aufgerufen werden:

[2]http://msdn.microsoft.com/en-us/library/windows/desktop/ms632600(v=vs.85).aspx; 27.08.2013

```
HWND hwnd_Knopf = CreateWindow (

"button",                       //Fensterklasse Knopf
"Das ist ein Knopf",            //Text auf den Knopf
WS_CHILD | WS_VISIBLE,          //Child Fenster und anzeigen
100,                            //Position in der X Koordinate des Hauptfensters
100,                            //Position in der Y Koordinate des Hauptfensters
50,                             //Lange des Knopfes
35,                             //Breite des Knopfes
hwnd_Fenster,                   //Zugehöriges Hauptfenster
(HMENU) ID_KNOPF,               //Identifikationsnummer, wird vorher deklariert
NULL,                           //keine Instanz
NULL                            //keine extra Information
);
```

Nachdem der Programmierer ein Hauptfenster erzeugt hat, muss dieses noch explizit mittels der Funktionen *ShowWindow(hwnd_Fenster, iCmdShow)* und *UpdateWindow(hwnd_Fenster)* angezeigt werden.

Die Message Loop

Windows hat eine Nachrichten Schleife (message loop) für jede laufende Anwendung. Wenn der Benutzer ein Event auslöst(Mausklick oder Tastatureingabe) muss das Programm reagieren. Windows übersetzt das Event in eine Nachricht und diese Nachricht wird in die Schleife reingeschrieben. Wenn eine neue Nachrichten empfangen wird, bevor die aktuelle Nachricht fertig bearbeitet wurde, wird die ankommende Nachricht in die „Message Queue“(Warteschlange) geschrieben. Jede Nachricht ist durch die *MSG* Struktur beschrieben.

```
typedef   struct tagMSG
{
          HWND        hwnd ;
          UINT        message ;
          WPARAM      wParam ;
          LPARAM      lParam ;
          DWORD       time ;
          POINT       pt ;
}
MSG,      * PMSG;
```

Jede Nachricht hat als erster Parameter ein Handle auf das zugehörige Fenster. Danach als *unsigned int (UINT)* wird die Nachricht übergeben. Diese Nachrichten sind in *WINUSER.H* deklariert und habe den Präfix *WM* für „Window Message“. Zum Beispiel wird ein Mausklick in das Fenster gemacht, schickt Windows eine Nachricht an das Fenster(durch *hwnd* angegeben) mit der Nachricht *WM_RBUTTONDOWN*.

wParam und *lParam* sind 32-Bit Nachrichtenparametern, abhängig vor der jeweiligen Nachricht. Wird zum Beispiel auf eine Knopf gedrückt, wird die *WM_COMMAND* Nachricht verschickt. Damit die Anwendung genau erkennen was für eine Aktion ausgeführt worden ist, wird

in *wParam* noch zwei weitere Angaben mitgeschickt. In die oberen 16-Bits wird die Notifikation *BN_CLICKED* verschickt. Dies deutet an, dass ein Knopf geklickt worden ist. In den unteren 16-Bits von *wParam* wird die Identifikationsnummer(nach den obigen Beispiel: ID_KNOPF) des Knopfes mitgeschickt. Mit diesen Informationen kann ein Programm genau auf die vom Benutzer ausgelöste Events reagieren. Beider 16-Bit Werte werden mit Hilfe der Makros *HIWORD* [3] und *LOWORD* [4].

Der Parameter *time* gibt die Uhrzeit wann die Nachricht verschickt worden ist und der Parameter *pt* ist eine *POINT* Struktur, wo die X und Y Koordinaten des Mausklicks gespeichert sind.

Eine Nachrichten Schleife ist standardmäßig so aufgebaut:

```
while   (GetMessage (&msg, NULL, 0, 0))
{
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
}
```

Die *GetMessage* Funktion speichert in der *msg* Struktur die aktuelle Nachricht. Die drei andere Parameter spezifizieren, dass alle Nachrichten(von der jeweilige Anwendung) in die Schleife geschrieben werden. Die Funktion liefert immer ein Wert ungleich Null, außer wenn die Nachricht *WM_QUIT* lautet, denn damit wird die Schleife verlassen und das Programm beendet.

TranslateMessage gibt die Nachricht an das Betriebssystem weiter und übersetzt die virtuelle Tasten Nachrichten auf tatsächliche Charakteren. Die *DispachtMessage* Funktion gibt die an Windows weiter. Gemeint ist, dass Windows die richtige Fensterprozedur (*WndProc* wird später erklärt)aufruft. Wenn die Fensterprozedur die Nachricht bearbeitet hat, gibt die Nachricht wieder an Windows. Windows gibt die bearbeitete Nachricht weiter an die jeweilige Anwendung und so kann die Schleife die nächste Nachricht laden und bearbeiten.

Fensterprozedur: WndProc Funktion

Unter der Initialisierung und Erzeugung der GUI wurde die Fensterprozedur schon einmal erwähnt. In dieser Variable ist der Name der Funktion, die die Nachrichten eines Fensters bearbeitet. Das heißt, wird eine Fensterstruktur als *WNDCLASS wc* deklariert, und der Parameter *WNDPROC lpfnWndProc* folgendermaßen initialisiert *wc.lpfnWndProc = WndProc;* muss die Fensterprozedur für dieses Fenster *WndProc* heißen. Wie die Fensterprozedur heißt ist irrelevant, solange die Namen in der Fensterklasse und die Prozedur übereinstimmen, somit sind Fenster und Prozedur verbunden. Eine Fensterprozedur ist immer so definiert:

```
LRESULT CALLBACK    WndProc(
                    HWND hwnd,
                    UINT message,
                    WPARAM wParam,
                    LPARAM lParam);
```

[3] http://msdn.microsoft.com/en-us/library/windows/desktop/ms632657(v=vs.85).aspx; 27.08.2013

[4] http://msdn.microsoft.com/en-us/library/windows/desktop/ms632659(v=vs.85).aspx; 27.08.2013

Der Datentyp *CALLBACK* ist schon bekannt, und *LRESULT* ist eine Typdefinition von *long*. Die vier Parameter der Funktion sind die gleichen wie die ersten vier Parameter bei eine *MSG* Struktur. Falls ein Programm mehrere Fenster von der gleichen Fensterklasse hat, spezifiziert das erste Parameter welches Fenster(Handle) die Nachricht schickt. Programme rufen die Fensterprozedur in der Regel nicht auf, sondern Windows. Will ein Programm die Prozedur direkt aufrufen, dann wird die Funktion *SendMessage*(wird später genauer erklärt) benutzt.

Der Sinn für eine von Programmierer geschriebene Fensterprozedur ist, dass ein Programm auf ein Event reagiert soll, wie der Programmierer es will. Programmierer programmieren in der Regel nicht alle Events. Damit eine Nachricht nicht unbearbeitet bleibt, wird immer am Ende der selbstgeschriebene Fensterprozedur die *DefWindowProc* Funktion aufgerufen. Diese Methode bearbeitet alle mögliche Nachrichten und ist die default Fensterprozedur. Wenn der Programmierer diese Funktion nicht aufruft, werden grundlegende Funktionen einer GUI nicht funktionieren.

Die erste Nachricht die eine Fensterprozedur bekommt ist *WM_CREATE* Nachricht. Hier baut der Programmierer alle GUI Elemente auf, mithilfe der *CreateWindow* Funktion(Knopf Beispiel). Eine zweite sehr wichtige Nachricht ist *WM_DESTROY*. Die Nachricht wird verschickt wenn der Benutzer ein Fenster zu machen will. Der Programmierer ruft die *PostQuitMessage(0)* Funktion auf und automatisch wird eine *WM_QUIT* Nachricht verschickt. Somit wird das Fenster geschlossen, die Message Loop verlassen und das Programm / Fenster richtig beendet.

Es gibt noch weitere Nachrichten, die der Programmierer „manuell“ verschicken kann. Mit Hilfe der erwähnte *SendMessage* [5] Funktion. Die Funktion hat die gleichen Parameter wie eine Fensterprozedur. Durch das Handle wird angegeben an welches Fenster die Nachricht verschickt werden soll. Durch das zweite Parameter wird angegeben, welche Nachricht verschickt wird, und die letzten zwei sind extra Informationen der Nachricht(wenn nötig). Will der Programmierer im Ablauf des Programms zum Beispiel der Darstellungstext von unseren Knopf ändern, wird die *SendMessage* Funktion so aufgerufen:

```
SendMessage(hwnd_Knopf, WM_SETTEXT, 0, (lParam)"NeuerText");
```

3.2.6 Die RS 232 Schnittstelle und die Windows API

Über die Windows API hat man direkt Zugriff auf die RS 232 Schnittstelle. Zum verwalten der Schnittstelle und die Eigenschaften zu setzen gibt es verschiedene Datenstrukturen die man aufrufen und ändern muss, je nach Bedarf.

Öffnen und Schließen eines Ports

Um Zugriff auf die Datenstrukturen zu haben, muss man zuerst eine RS 232 Schnittstelle (COM Port) öffnen. Durch die Funktion CreateFile[6] bekommt man ein Handle auf den angegeben Port. Ein Handle ist eine Referenzwert zu einer vom Betriebssystem verwalteten Systemressource, im diesem Fall eine im System vorhandene RS 232 Schnittstelle. Mit *CreateFile* kann man auch Zugriff auf Dateien, Datenstreams und andere Kommunikationsressourcen bekommen. Das Handle muss gespeichert werden, denn damit wird der jeweiliger Port identifiziert und angesprochen für

[5] http://msdn.microsoft.com/en-us/library/windows/desktop/ms644950(v=vs.85).aspx; 28.08.2013

[6] http://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx; 25.08.2013

weitere Operationen. Durch die *CreateFile* Funktion wird die Datei, oder im diesem Fall die Input / Output Schnittstelle für diese Anwendung reserviert. Das heißt, für das Betriebssystem und andere Anwendungen steht diese Schnittstelle nicht mehr zur Verfügung.

Um den richtigen Zugriff auf einen Port zu haben, muss auch die richtige Flags bei dem Aufruf der *CreateFile* Funktion angegeben werden. Als Flags sind die folgende Parameter anzugeben:

- Schreib und Leserechte
- Non-Sharing Modus
- Öffnen vor nur existierende Schnittstellen
- Asynchron Modus

Um ein Programm „sauber“ zu beenden müssen offene Handles geschlossen werden. Durch die Funktion *CloseHandle*[7] mit Angabe eines gültiges Handles wird dieses geschlossen, und steht für das Betriebssystem und andere Anwendungen wieder zur Verfügung.

Konfiguration eines Ports

Es gibt drei wichtige Strukturen, die für die Konfiguration einer COM Schnittstelle relevant sind. Mittels dieser Strukturen werden die Eigenschaften, einstellbare Parameter und Wartezeiten einer Schnittstelle eingestellt.

Die *DCB*[8] Struktur definiert die Ansteuerungseigenschaften für die Schnittstelle. Um die Struktur für eine angegebene Schnittstelle zu laden, wird die Funktion *GetCommState* aufgerufen. Diese Struktur besteht aus 28 verschiedene Variablen die ich nicht einzeln benennen werde. Diese Variablen beschreiben wie die Schnittstelle konfiguriert ist. Man kann die Baudrate, Parität, Stopbits, Datenbits, Flusssteuerung und Fehlerbenachrichtigung einstellen.

Um die Schnittstelle richtig einzustellen und keine falsche Eingaben in die *DCB* Struktur zu schreiben, ist es sinnvoll vorher die *COMMPROP* [9] Struktur zu auswerten. Durch die Funktion *GetCommProperties* wird die Struktur für die angegebene Schnittstelle geladen. Diese Struktur besteht aus 18 Variablen und beschreibt die mögliche Einstellung für die Schnittstelle. Die Einstellungen werden aus dem Treiber der jeweiliger Schnittstelle(Onboard Ports, Sunix, Intel SOL oder ITE) gelesen. Die für diese Arbeit relevanter Parameter ist die maximale einstellbare Baudrate.

Während der Übertragung von Daten über eine Serielleschnittstelle sind maximale Wartezeiten fällig. Dieses Werte entstehen durch die Übertragungslänge und die eingestellte Baudrate. Auch die Wartezeit zwischen zwei ankommende Bytes ist wichtig um das Ende der Übertragung zu bestimmen. Diese Werte sollten dynamisch berechnet werden und danach die Schnittstelle einstelle. Dafür ist die *COMMTIMEOUTS* [10] Struktur zuständig. Die Struktur besteht aus fünf Variablen

[7] http://msdn.microsoft.com/en-us/library/windows/desktop/ms724211(v=vs.85).aspx; 25.08.2013

[8] http://msdn.microsoft.com/en-us/library/windows/desktop/aa363214(v=vs.85).aspx; 28.08.2013

[9] http://msdn.microsoft.com/en-us/library/windows/desktop/aa363189(v=vs.85).aspx; 28.08.2013

[10] http://msdn.microsoft.com/en-us/library/windows/desktop/aa363190(v=vs.85).aspx; 28.08.2013

die die Wartezeiten in Millisekunden angeben. Die erste Variable beschreibt die Zeitüberschreitung zwischen zwei ankommende Bytes, wird dieser Wert überschritten, so wird die Leseoperation beendet.

Der zweite und vierte Parameter(jeweils für Lesen und Schreiben) beschreiben die Zeit, die für die Übertragung aller Bytes nötig ist(für die aktuelle Schreibe- oder Leseoperation). Diese Parametern sind von der Baudrate und die Bitlänge eines Charakters. Damit ist gemeint, wie unter 3.1.2 erläutert, ein Charakter kann aus 8 Bits plus Startbit, Stopbit und Parität bestehen. Also werden nicht genau 8 Bits(1 Byte) per Charakter, sondern oft mehr (11 Bits zum Beispiel) übertragen. Die Wartezeit wird durch die folgende Formel berechnet.

$$Wartezeit = \frac{AnzahlderBits}{Baudrate} \times 1.1$$

In der Formel wird mal 1.1 multipliziert um eine 10% Kulanz zu haben. Der dritte und fünfte Parameter sind eine extra Wartezeit, für jeweils Lesen und Schreiben, die zu der gesamte berechnete Wartezeit pro Lese-/Schreibvorgang addiert wird.

Lesen und Schreiben

Events and Interrupts

Kapitel 4

Lösungsansätze

Nach der Beschreibung des fachlichen Umfeldes können ausgewählte (vielleicht alle?) für die Aufgabenstellung relevanten Probleme vorgestellt, Vor- und Nachteile bestehender Lösungen argumentiert und die voraussichtlich angestrebte (weil vorteilhafte) Lösung herausgestellt werden.

4.0.7 Fehlererkennung

4.0.8 Master Slave Modus zwischen zwei Systeme

problem -> erzeugung eines protokolls, master ready and slave ready? timeouts..... recon events 2.7.5 Synchronisierungsprotokoll

4.0.9 Challenges of C++ and the gui

template class

Kapitel 5

Anforderungsdefinition

Hier werden die Grundlagen für das zu entwickelnde Softwaresystem definiert. Zwar noch aus fachtechnischer Sicht werden hier die Anforderungen an das geplante Softwaresystem in möglichst formaler Form spezifiziert. Es sollen hier keine Lösungen präsentiert werden, sondern möglichst präzise die Anforderungen (Sollkonzept) an das geplante Softwaresystem mit seinen Schnittstellen, Informationsflüssen und Systemfunktionen dokumentiert werden. Verwendete Methoden können z.B. SA, SADT, Petri-Netze oder andere sein. Das Ergebnis ist ein für die Systementwicklung verwendbares Pflichtenheft. Über Art und Umfang des Pflichtenhefts sollten Sie mit Ihrem Betreuer sprechen.

Kapitel 6

Systementwurf

Auf der Basis des Pflichtenhefts werden aus softwaretechnischer Sicht die Anforderungen an das System spezifiziert. Hierzu gehört minimal eine Beschreibung auf höherem Niveau (Modulebene), eine auf mittlerem Niveau (Struktogramme, Pseudocode oder Spezifikation von Prozeduren (Funktionen) sowie die Beschreibung der für das System essentiellen Datenstrukturen (z.B. als Datenlexikon). Typische Beschreibungen sind die Modulhierarchie (oder Modulgraph), eine Spezifikation aller Module mit ihren Schnittstellen (inklusive Zweck, Ein-/Ausgabe), sowie eine Spezifikation aller in den Modulschnittstellen liegenden Prozeduren und Funktionen. Bestandteil des Entwurfs sollten nicht nur die jeweiligen Ergebnisse, sondern auch die Beschreibung des Entwicklungsweges (inklusive verworfener Lösungen) sein.

Kapitel 7

Realisierung

Selbstverständlich sollte Realisierung auch in Ihrer Arbeit abgehandelt werden. Aus der Sicht der Softwaretechnik stellt sie aber nur der kronende Abschluß der Arbeit dar. Hier können die realisierungsspezifischen Probleme (z.B. mit der Implementierungssprache) und das Testkonzept inkl. der protokollierten Testergebnisse dargestellt werden. Wichtige, komplexe oder besonders interessante Systemteile können auch im Programmcode dargestellt werden. Bitte aber Hinweis 12 (Programm-Listings) beachten!

```
1
2 HANDLE WINAPI CreateFile(
3   _In_      LPCTSTR lpFileName,
4   _In_      DWORD dwDesiredAccess,
5   _In_      DWORD dwShareMode,
6   _In_opt_  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
7   _In_      DWORD dwCreationDisposition,
8   _In_      DWORD dwFlagsAndAttributes,
9   _In_opt_  HANDLE hTemplateFile
10 );
```

Kapitel 8

Bedienungsanleitung

Wie läßt sich das entwickelte System bedienen? Welche Fehlermeldungen existieren? Was sind die Ursachen? Wie muß der Benutzer auf auftretende Bedienungs- oder Systemfehler reagieren? Bedienungsanleitungen sollten auf den Kenntnisstand und Sprachgebrauch des zukünftigen Anwenders zugeschnitten sein. In der Form und Aufmachung können SIe ein Aushängeschild für Ihre Arbeit sein. Übersichtsdarstellungen wir Menübäume, Befehlslisten und/oder Systaxdiagramme erleichtern die Benutzung eines Systems umgemein und gehören heute zum Standard guter Dokumentationen. Neben der Einzeldarstellung von Befehlen und Funktionen können die Aufzeichnung von Beispielsitzungen, Fallbeispielen das Verständnis für das System wesentlich vertiefen. (Anmerkung: Je nach Art des entwickelten Systems ist es manchmal sinnvoll die Bedienungsanleitung in ein größeres Kapitel Anwenderdokumentation einzubetten.)

Kapitel 9

Zusammenfassung und Ausblick

Lehnen Sie sich zurück von Ihrem Terminal und versuchen ein wenig Abstand zu den vielen Detail-Problemen Ihrer Diplomarbeit zu gewinnen: Was war wirklich wichtig bei der Arbeit? Wie sieht das Ergebnis aus? Wie schätzen Sie das Ergebnis ein? Gab es Randbedingungen, Ereignisse, die die Arbeit wesentlich beeinflußt haben? Gibt es noch offene Probleme? Wie könnten diese vermutlich gelöst werden?

Abbildungsverzeichnis

Listings

Anhang A

Anhang

HIER KOMMEN DIE DATEIANHÄNGE HIN