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
25. August 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 zu programmierende Programm soll in der Programmiersprache „C / C++“ entwickeltet werden und auf folgenden Betriebssystemen implementiert werden:

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

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

Das Programm soll die Schwächen anderer Serial Ports Tools ergänzen, wie zum Beispiel die eingeschränkte Skriptfähigkeit. Diese Tools unterstützen keine Zähler, keine Uhrzeit/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, weil die Wincor Nixdorf Kunden und 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 Nutzung der Windows API die Installation oder Implementierung des Tools auf einem Linux-System nicht möglich.

Das Programm soll aus einer GUI bestehen, in der alle Programm- und Schnittstellenparameter (Baudrate, Stoppbits, Paritätsbit, Datenbits, Hardware RTS/CTS, Hardware DTS/DSR, Software XON/XOFF) eingestellt werden kann. Wenn das Programm über die Kommandozeile oder ein Skript gestartet wird, muss eine Konfigurationsdatei oder die Konfigurationsparameter übergeben werden.

Kapitel 2

Aufgabenstellung

Das Tool soll die RS 232 Schnittstelle testen und die Schwächen der vorhandenden Lösung ergänzen. Wincor Nixdorf besitzt schon ein Tool was die Druckertestumgebung testet, dies ist aber Ruby basierend. Die vorhandenen Lösung, wie zum Beispiel ÄGGsoft Serial Port Tester"haben eingeschränkte Einstellungen und sind nicht ausführlich genug für die Voraussetzungen der Qualitätsicherung.

AGGsoft Serial Port Monitor

Unterschied zu / Mängel vom AGGsoft Serial Port Monitor? a) Eingeschränkte Skriptfähigkeit: Zähler, Uhrzeit/Datum als Sendedaten, kein Zufallsgenerator ... b) Win PE?

WN Druckertestumgebung

(Ruby basierend) Setzt nicht direkt auf der gleichen Schnittstelle (Win-API) auf, wie Kundenapplikationen, also ungeeignet.

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 OK und/oder was dafür noch 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, z.B.:

<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 Syntaxprüfung.

2.9.2 Kommandozeile

Die Kommandozeile nimmt folgende Optionen entgegen:

- /Master oder /Slave oder /Full
-
-
-
-

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

http://de.wikipedia.org/wiki/RS-232 Joe Campbell: V 24 / RS-232 Kommunikation. (6313 736). 4. Auflage. Sybex-Verlag GmbH, 1984, ISBN 978-3-8874-5075-5.

3.1.1 Definition und Geschichte

3.1.2 Verkabelung und Stecker

3.1.3 RS 232 in Wincor Nixdorf

stromversorgung ba66 kassenlade

3.2 Microsoft Windows API

3.2.1 Definition

Die Quellen des folgenden Kapitel sind aus "Programming Windows, Charles Petzold, 1995ünd "Visual C++ 2010, Dirk Louis, 2010".

Die Microsoft Windows Äpplication 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 Editionünd 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 *ïnt"* "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 *ïnt"* 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

HEADERS
DEFINES
ETC

3.2.3 GUI

Durch die Hardware und die Mitte der 70 Jahren wurde in Xerox Palo Alto Research Center graphische Benutzeroberflächen recherchiert. Aus diesen Ergebnisse profitierten Macintosh und Windows, und bauten darauf Ihre graphische Benutzeroberflächen. Für dieses Projekt ist gefordert, ein Tools mit gleichgewichtige Kommandozeile und GUI. In diesen Zeiten der Computerwelt kann man sich als Benutzer kaum eine Welt ohne Benutzeroberflächen vorstellen. Die Windows API hat seit der Ankündigung(1983) und Veröffentlichung(1985) von Windows als Betriebssystem Funktionsaufrufe für die Programmierung von GUI's. Die Benutzung der API für die Programmierung von GUI 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.

3.2.4 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[1] 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 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[2] mit Angabe eines gültiges Handles wird dieses geschlossen, und steht für das Betriebssystem und andere Anwendungen wieder zur Verfügung.

Konfiguration eines Ports

DCB COMMTIMEOUTS COMMPROP

Lesen und Schreiben

Events and Interrupts

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

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

Kapitel 4

Lösungsansätze

hgdjkgfjkshgfkhjsagjhlfhaklshfklafjkajksfhjkadshjklfgadjklfga sadfasjhdfghjkasgdhjkfgahjksdgfh-
jasgjlhfgajsfjlhasd SDGFSAJKFGJKHASDGFHJKASGHJKFGAHJKSDFGHJASGHDF .asdh-
jgshjkafghjkasfcachvlkwklr tekluwthkerhköaenkfgae

Kapitel 5

Anforderungsdefinition

hgdjkgfjkshgfkhjsagjhlfhaklshfklafjkajksfhjkadshjklfgadjklfga sadfasjhdfghjkasgdhjkfgahjksdgfh-
jasgjlhfgajsfjlhasd SDGFSAJKFGJKHASDGFHJKASGHJKFGAHJKSDFGHJASGHDF .asdh-
jgshjkafghjkasfcachvlkwklr tekluwthkerhköaenkfgae

Kapitel 6

Systementwurf

hgdjkgfjkshgfkhjsagjhlfhaklshfklafjkajksfhjkadshjklfgadjklfga sadfasjhdfghjkasgdhjkfgahjksdgfh-
jasgjlhfgajsfjlhasd SDGFSAJKFGJKHASDGFHJKASGHJKFGAHJKSDFGHJASGHDF .asdh-
jgshjkafghjkasfcachvlkwklr tekluwthkerhköaenkfgae

Kapitel 7

Realiseriung

hgdjkgfjkshgfkhjsagjhlfhaklshfklafjkajksfhjkadshjklfgadjklfga sadfasjhdfghjkasgdhjkfgahjksdgfhjasgjlhfgajsfjlhasd SDGFSAJKFGJKHASDGFHJKASGHJKFGAHJKSDFGHJASGHDF .asdhjgshjkafghjkasfcachvlkwklr tekluwthkerhköaenkfgae

```
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

hgdjkgfjkshgfkhjsagjhlfhaklshfklafjkajksfhjkadshjklfgadjklfga sadfasjhdfghjkasgdhjkfgahjksdgfh-
jasgjlhfgajsfjlhasd SDGFSAJKFGJKHASDGFHJKASGHJKFGAHJKSDFGHJASGHDF .asdh-
jgshjkafghjkasfcachvlkwklr tekluwthkerhköaenkfgae

Kapitel 9

Zusammenfassung und Ausblick

hgdjkgfjkshgfkhjsagjhlfhaklshfklafjkajksfhjkadshjklfgadjklfga sadfasjhdfghjkasgdhjkfgahjksdgfh-
jasgjlhfgajsfjlhasd SDGFSAJKFGJKHASDGFHJKASGHJKFGAHJKSDFGHJASGHDF .asdh-
jgshjkafghjkasfcachvlkwklr tekluwthkerhköaenkfgae

Abbildungsverzeichnis

Listings

Anhang A

Anhang

HIER KOMMEN DIE DATEIANHÄNGE HIN