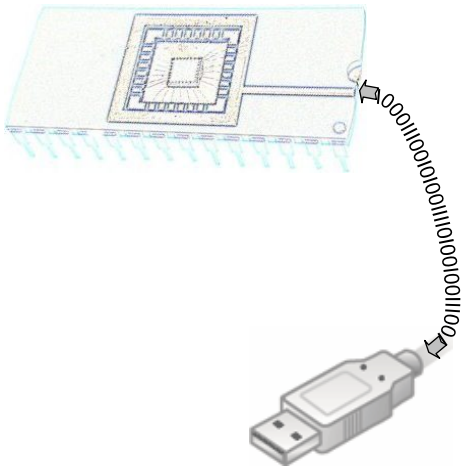
FH Karlsruhe
UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE FÜR TECHNIK

Projektarbeit
USB 2.0

Energie- und Automatisierungstechnik

WS 04/05



Projektleiter:

Prof.Dr.-Ing. H. R. Fehrenbach

Bearbeiter:

- Bauer, Fabian
- Hardt, Arthur
- Lopez, Alexander
- Schubert, Christian
- Shamshoum, Wael
- Ziegler, Steffen

INHALTSVERZEICHNIS

1. Einführung

1.1 Aufgabenstellung

Thema: USB2

Zur Realisierung von universellen, kostengünstigen und schnellen I/O-Schnittstellen hat sich der USB-Bus hervorragend bewährt. USB2 bietet darüber hinaus hohe Datentransferraten. Inzwischen gibt es Interface-Bausteine mit angekoppeltem 8051-kompatibelem Mikrokontroller in Form von Prototypenplatinen (siehe http://braintechnology.de/braintechnology/product.html).
Für die Datenkommunikation steht eine freie (LGPL), plattformübergreifende Programmbibliothek zur Verfügung (siehe http://libusb.sourceforge.net oder http://libusb-win32.sourceforge.net).
Zur Programmierung des 8051-kompatiblen Mikrocontrollers kann ein freier C-Compiler (siehe http://sdcc.sourceforge.net) benutzt werden.
Es soll eine USB2-Prototypenplatine beschaffen und in Betrieb genommen werden. Mit Hilfe eines Testaufbaus sollen erreichbare Datentransferraten und Latenzzeiten bestimmt werden.
Weiter Informationen unter:
http://atlas.uni-wuppertal.de/~dopke/wodan2/ezusb.html erhältlich.

Aufgaben:

- Beschaffung und Inbetriebnahme der Prototypenplatine
- Inbetriebnahme des freien C-Compilers für 8051-Mikrocontroller
- Inbetriebnahme und Test der Programmbibliothek „libusb“
- Entwurf und Realisierung eines Testaufbaus für die Ermittlung der erreichbaren I/O-Leistung

Anzahl der Bearbeiter: 6

1.2 Erste Schritte

Aufgaben: Es werden kleinere Arbeitsgruppen gebildet um schnell und effizient das Projekt voran zu bringen.

- Hardware
- Software
- Diagnose/Test
- Dokumentation

Arbeitsumfeld: Als Projektraum steht uns das Labor LiU09 zur Verfügung. Diverse Rechner, Arbeitsunterlagen, Werkzeuge, etc. können dort hinterlassen werden und ein zentraler Treffpunkt ist somit vorhanden.

Der Raum ist für unautorisierte Personen nicht zugänglich.
Eigene Schlüssel können im Rektorat beantragt werden.

Arbeitsmittel: Uns werden diverse Rechner der FH mit Internetzugang zur Verfügung gestellt. Ansprechpartner hierfür ist Herr Gantner (Li028), der für die IT-Systeme zuständig ist.
Ein Laserdrucker befindet sich im Projektraum.
Messgeräte und sonstige Hilfsmittel (Kabel, Werkzeuge, etc.) können bei Herrn Wäldle (Li021) ausgeliehen werden.

Zielsetzung: Das Ziel ist möglichst schnell ein Vorgehensweise auszuarbeiten, um in dem vorgegebenen Zeitraum zu einem positiven Ergebnis zu gelangen.

Projektname: Also Projektnamen wählen wir PROTEuS (vgl. griech. Mythologie) welcher die Flexibilität und Anpassungsfähigkeit der Platine an verschiedenste praktische Anwendungen betonen soll.

Zeitplan: Um den zeitlichen Verlauf des Projektes zu dokumentieren wird ein Projektplan mit dem Meilenstein-Verfahren erstellt. Dieser definiert die einzelnen Daten der Teilaufgaben sowie den möglichen Abschlusstermin. Als grobes Abschlussdatum haben wir den 22. März 2005 gesetzt welches jedoch ein unrealistisches Datum ist im Bezug auf die diversen Komplikationen und Probleme welche mit Fortschritt des Projektes entstanden (vgl. Zeitplan im Anhang).

Arbeitshergang: Das Arbeitslabor (LiU09) dient zur Forschung und Entwicklung am USB-Interface. Zur Auswertung des Projektverlaufes wird eine wöchentliche Statusrunde mit allen Beteiligten (donnerstagnachmittags) festgelegt. Im Projektraum wird die meiste Zeit investiert um Fortschritte in Bezug auf Softwareprogrammierung und Hardwareentwicklung zu erzielen. Zur Festhaltung der Ergebnisse haben wir ein Protokoll der jeweiligen Sitzung erstellt (siehe Projektprotokolle im Anhang).

Kommunikation: Da der Austausch von aktuellen Informationen und Problemen eine wichtige Grundlage im Bezug auf eine effiziente Arbeitsweise ist, wird ein e-Mail-Verteiler eingerichtet (Projektmitglieder: usb@stenox.de, Projektmitglieder + Herr Fehrenbach: usb2@stenox.de) der schnelle und gezielte Verbreitung von Daten zwischen den Teilnehmern sicherstellt.

Archivierung: Alle Projektdaten werden auf einer Backup-CD gesammelt um mögliche Verluste durch Rechnerabstürze etc. zu vermeiden. Auch über eine eigene Webseite (http://www.home.fh-karlsruhe.de/~bafa0012/usb2/index.html) und wichtige Informationen und Daten archiviert und für alle Teilnehmer zugänglich.

2. Informationsrecherche

2.1 USB 2.0

Der **Universal Serial Bus** (**USB**) ist ein Bussystem zur Verbindung von einem Computer mit externen Geräten zum Austausch von Daten. Durch die relativ hohen möglichen Datenraten und die automatische Erkennung von Geräten und deren Eigenschaften ist der USB zum Anschluss fast aller Gerätearten von Maus und Tastatur bis zu Festplatten und Kameras vorgesehen. Die Anzahl der USB-Anschlüsse eines Computers kann mit USB-Hubs vergrößert werden.

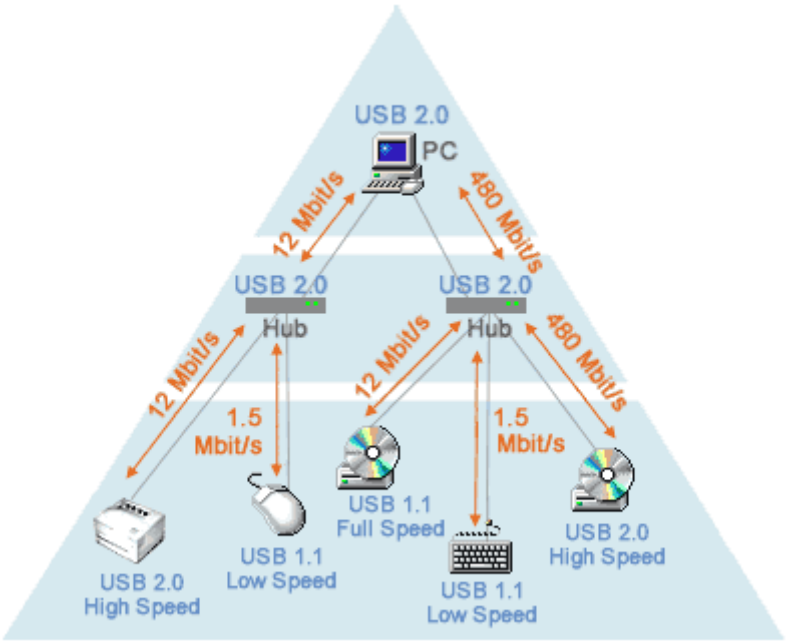
USB ist ein serieller Bus. Das bedeutet, das Datenkommunikation auf lediglich einer differenziellen Datenleitung (zwei Leitungen, eine überträgt invertierte Daten - dadurch reduziert sich die Abstrahlung und erhöht sich die Übertragungssicherheit) erfolgt. Dabei werden die einzelnen Bits des Datenpaketes nacheinander übertragen. Durch Verwendung nur einer Datenleitung können die Kabel dünner und preiswerter als bei parallelen Schnittstellen ausgeführt werden und die hohe Datenrate ist mit relativ geringem Aufwand zu erreichen, da nicht mehrere Signale mit identischem elektrischem und zeitlichem Verhalten übertragen werden müssen.

Der Bus erlaubt es, bis zu 127 verschiedene Geräte an einem Hostcontroller anzuschließen. Durch Verwendung mehrerer Hostcontroller können auch noch mehr Geräte angeschlossen werden, jedoch ist dies abhängig vom verwendeten Betriebssystem, Windows beispielsweise wird lange vor Erreichen der 127 Geräte instabil. USB zeichnet sich dadurch aus, dass die Installation der Geräte verhältnismäßig einfach ist und die Datenkabel der Geräte im laufenden Betrieb ein- und ausgesteckt werden können.

Gegenüber den bisherigen externen Schnittstellen am PC bietet USB deutlich höhere Datenübertragungsraten. USB stellt aus all diesen Gründen eine gute Weiterentwicklung zu den bisher verwendeten Schnittstellen (parallel/Centronics und seriell/RS232) dar.

Die aktuelle Version der USB Spezifikation ist 2.0. Mit dem Schritt von der V1.1 zur 2.0 wurde es ermöglicht Geräte mit einer deutlich höheren Datenrate zu bauen.

Zu beachten ist, dass es bei USB (im Gegensatz etwa zu RS-232) zwei Arten von Controllern gibt: Host- und Devicecontroller. Host bezeichnet dabei die steuernde Seite und ist z.B. in PCs zu finden; Devices sind die USB-Geräte, z.B. USB-Webcams. Diese Unterscheidung ist wichtig, da die meisten USB-Lösungen für Mikrocontroller USB-Devices darstellen und man deswegen dort weder Webcams noch USB-Speichersticks anschließen kann. Mit der letzten Ergänzung des Standards (USB On-The-Go) gibt es die begrenzte Möglichkeit, dass Geräte Host-Funktionalität zur Kommunikation mit ausgewählten Peripheriegeräten erhalten.



Datenübertragungsraten der USB-Standards

Eckdaten	USB 1.1 Low Speed	USB 1.1 Full Speed	USB 2.0 High Speed
Übertragungsrate	1.5 MBit/s	12 MBit/s	480 MBit/s
Max. Endpunkte	2	31	31
Max. Bulk-Paketgröße	8 Byte	64 Byte	512 Byte
Max. Bulk-Übertragungsrate	16 KByte/s	1,1 MByte/s	56 MByte/s
Max. Isochrone Paketgröße	nicht möglich	1023 Byte	1024 Byte
Max. Isochrone Übertragungsrate	nicht möglich	1 MByte/s	24 MByte/s

2.2 USB-Kabel

In einem USB-Kabel werden nur vier Adern benötigt: zwei für die Spannungsversorgung von 5V (bei maximal 500 mA sprich 2,5W) und zwei für die Datenübertragung.

Die Kabel müssen je nach Geschwindigkeit unterschiedlich geschirmt werden. Nur Kabel, die der Spezifikation für Full Speed entsprechen dürfen mit A und B Stecker versehen sein. Low Speed Kabel müssen generell als "captive cable", also fest angeschlossenes Kabel ausgeführt sein (die Verwendung eines eigenen Steckers auf Geräteseite ist möglich, es darf nur kein USB Stecker sein), da durch die geringe Abschirmung Probleme auftreten würden wenn dieses Kabel mit einem Full Speed Gerät benutzt wird.

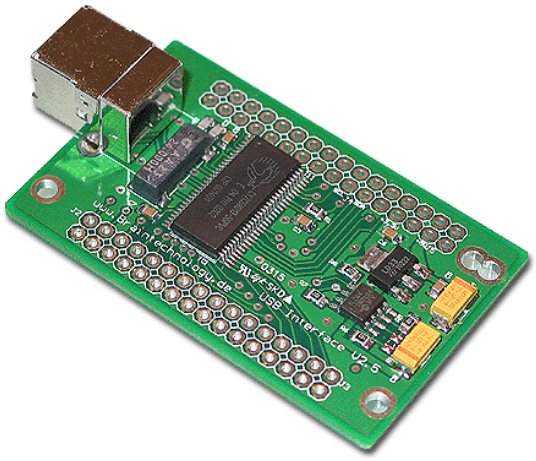
Ein High Speed Kabel gibt es nicht, die Full Speed Kabel der USB 1.0 und 1.1 Spezifikation waren technisch bereits völlig ausreichend um die 480 MBit/sec des High Speed USB zu erlauben. Kabel, die im Handel als "USB 2.0 Kabel" verkauft werden, sind daher als Aufschneiderei anzusehen.

Die Länge eines Kabels vom Hub zum Device ist auf fünf Meter begrenzt (Low Speed Kabel werden von der Spezifikation auf 3 m beschränkt, dies ist technisch jedoch unbegründet und wird voraussichtlich in einer zukünftigen Fassung der Spec entfallen). Die Spezifikation schließt Verlängerungen aus. Sind sie notwendig, müssen USB-Hubs dazwischen geschaltet werden. Diese und andere Geräte mit geringem Stromverbrauch können über den Bus mitversorgt werden. Einer der Vorteile geringer Adernzahl sind kleine Stecker, die zudem verpolungssicher ausgeführt sind.

USB-Stecker Typ A

USB-Stecker Typ B

2.3 USB High Speed Interface V2.5 mit I²C EEPROM



- USB 2.0 und 1.1 Kompatibel
- für High Speed 480Mbs USB Bus
- kompatibel zu Full Speed 12Mbs USB Bus
- 5V / 3 V Betriebsspannung bei ca. 30mA
- I²C Bus 100Khz/400Khz
- Smart Serial Interface
- GPIF Interface
- Bulk/Interrupt/Isochronous
- Parallel Bus
- sehr Kompakt
- EZ-USB FX2 Kompatibel (Cypress)
- 8k (CY7C68013) Programmspeicher
- I²C EEPROM 8 KB
- Alle Anschlüsse herausgeführt
- Pin kompatibel zu USB Interface V1.2/V1.3
- Spannungsregler 3,3V ca. 400mA

Beschreibung:
Interface für High Speed USB Bus Anwendungen. Der Anwender hat sich nicht um das komplizierte USB BUS Protokoll zu kümmern, sondern kann direkt mit dem USB Bus arbeiten. Eine **USB2.dll** (kostenpflichtige, Demo Version unter www.braintechnology.de verfügbar) wird in die Entwicklungsumgebung eingebunden. Der Anwender kann dann über Funktionen der USB2.dll auf seine Hardware zugreifen.
Die Software der Interfacekarte wird über die USB2.dll automatisch in den Baustein geschrieben, so dass immer das neuste Update der Software im Interface (Hardware) installiert ist.
Das Interface hat eine I²C Schnittstelle und eine parallele Schnittstelle mit Daten und Adressleitungen. Mit diesem Interface können Sie also sofort mit USB loslegen!
Natürlich kann der Controller oder das EEPROM mit eigener Software beschrieben werden, so dass die Platine als sehr gutes Entwicklungskit genutzt werden kann.

Anwendung:

- Interface für Entwicklung am USB-Bus
- ATA IDE Interface
- Schnittstellenadapter für USB zu Parallel und I^2C Bus
- Entwicklungskit für eigene Anwendungen
- Prototypenentwicklung

2.4 Der 8051 Prozessor

Die **8051**-Familie (besser: MCS-51) ist eine Prozessorarchitektur von Intel. Der Original 8051 ist mittlerweile veraltet, aber es gibt hunderte Varianten (Derivate) davon, die teilweise durchaus auf dem aktuellen Stand der Technik sind.

Kenndaten des Original 8051:

- jeweils bis zu 64kB externer Daten- und Programmspeicher adressierbar
- 128 Byte internes RAM (8052: 256 Byte)
- 2 Timer/Counter (8052: 3 Timer/Counter)
- 2 externe Interrupts
- 4 8-bit I/O Ports, zwei davon für den Zugriff auf externen Speicher
- Hardware UART (**U**niversal **A**synchronous **R**eceiver **T**ransmitter)

Der Original 8051 ist ein maskenprogrammierter Mikrocontroller; die ROM-lose Variante heißt 8031. Für einen Befehl benötigt er mindestens 12 Takte. Befehls- und Datenspeicher sind logisch getrennt, auch wenn diese über einen einzigen gemultiplexten externen Bus adressiert werden - sofern externe Speicher verwendet werden. Ob es sich hierbei um eine Harvard-Architektur oder eine Von Neumann-Architektur handelt, ist umstritten. In der Standardbeschaltung ist kein Code im Datenspeicher als Programmcode ausführbar. Dieses lässt sich jedoch durch Verschalten von PSEN und RD per AND erreichen. Diese Methode wird während der Programmentwicklung gerne verwendet.

Modernere 8051-Derivate warten unter anderem neben höherer Taktfrequenz, I²C-Bus, DA-Wandler und AD-Wandler mit einer geringeren Taktteilung auf. Dadurch werden zur Ausführung eines Befehls nicht mehr 12, sondern nur noch 6 oder sogar nur 4 Takte benötigt. Durch einen internen Flash-ROM als Programmspeicher entfällt die Notwendigkeit ein externes EPROM anzuschließen. Darüber hinaus verwenden einige USB-Mikrocontroller einen 8051-Kern, beispielsweise die EzUSB von Cypress oder die TUSBxxxx-Serie von Texas Instruments

2.5 I²C-BUS

Der **I²C** ist ein synchroner serieller Zweidraht-Bus (eine Daten- und eine Taktleitung), welcher für die Kommunikation zwischen ICs über kleine Distanzen geeignet ist. Entwickelt wurde er Anfang der 80er Jahre von Philips. Gesprochen "I quadrat C" steht für IIC = Inter Integrated Circuit Bus. Aus Lizenzgründen heißt der I²C Bus bei manchen Herstellern auch **TWI (T**wo **W**ire **I**nterface).

In einem I²C-Bus gibt es mindestens einen Master und eine beliebige Anzahl Slaves (max. 128). Ein I²C-Bus mit mehreren Mastern wird als "Multi-Master-Bus" bezeichnet. Der (oder die) Master sprechen die Slaves an; ein Slave kann NIE selbstständig Daten senden. Dazu übernimmt der Master, der Daten senden oder empfangen möchte, den Bus und gibt die (7-bit- bzw. 10-bit-) Adresse des Slaves aus, mit dem er kommunizieren möchte.

Nach der Adresse teilt der Master dem entsprechenden Slave mit, ob er Daten senden oder empfangen möchte. Danach werden die eigentlichen Daten (entweder vom Master oder Slave) auf den Bus gelegt. Hat der Master den Lese- oder Schreibvorgang abgeschlossen, so gibt er den Bus wieder frei. Sofern mehrere Master vorhanden sind, stellt ein Protokoll sicher, dass sich diese nicht gegenseitig stören.

Die Übertragungsrate beträgt beim Standard mode bis zu 100 kbit/s, beim Fast mode bis zu 400 kbit/s und beim High-speed mode bis zu 3,4 MBit/s. Falls die Taktrate für einen Slave zu hoch ist, kann er die Clock-Leitung auf Null ziehen und die Übertragung damit verlangsamen. Dies ist auf Bit- wie auf Byte-Ebene möglich, Ersteres allerdings nicht im High-speed mode.

3. Hardware

3.1 I/O_Testplatine I

Um eine einfache Praktische Anwendung des Interface zu demonstrieren fertigten wir eine Testplatine auf der einzelne Signale eingestellt, ausgelesen und wieder ausgegeben werden konnten. Rote Leuchtdioden dienten zur optischen Erkennbarkeit einer Signalausgabe.

3.2 Funktionsweise

- **IN-Ports:**
  Die Eingänge werden mittels 8 Dipp-Schalter (PD0 - PD7) gesteuert. Hier können verschiedene Bitmuster eingestellt werden welche das Interface dann einliest und zur Weiterverarbeitung Speichert.

- **OUT-Ports:**
  Um die Zustände (High-Low oder 1 und 0) des Interfaces darzustellen, wurden zur optischen Verdeutlichung den Output-Ports (PA0 – PB7 und PB0 – PB7) LED's vorgeschaltet. Diese werden mit der 5 Volt Betriebsspannung des PC's versorgt. Um eine Überlastung der LED's zu vermeiden sind diesen jeweils 330Ω Widerstände (R_V) vorgeschaltet. Dabei ist zu beachten, dass es zu keiner Überlastung des USB-Ports kommen darf, da die gängige Belastung bei 2,5 Watt liegt.

$$R_V = \frac{U_b - U_{LED}}{I_{LED}} = \frac{5V - 1{,}7V}{10mA} = 330\Omega$$

- **Reset:**
  Durch Betätigung des Reset-Knopfes, wird das Interface in seinen Ursprungszustand zurückgesetzt.

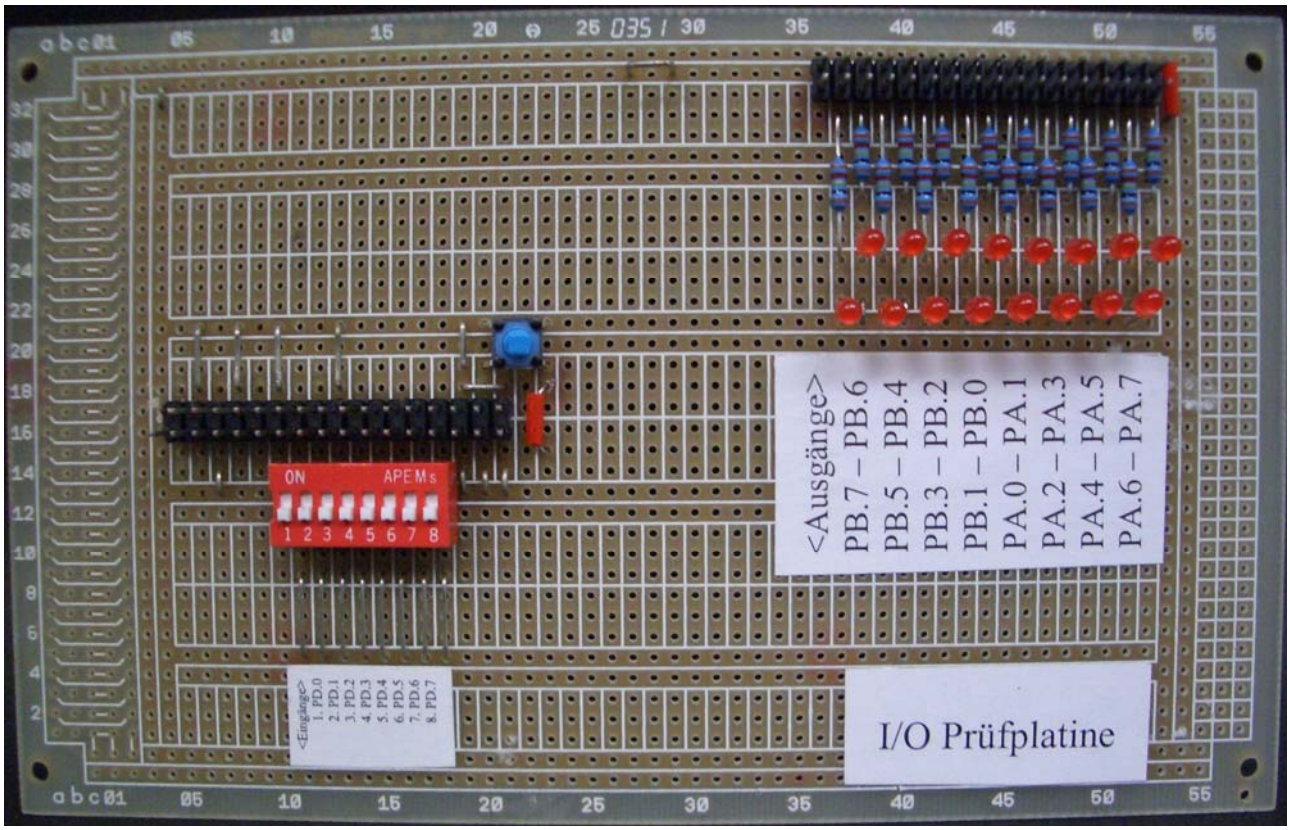
3.3 Aufbau

Die Flachbandkabel sind 34-Polig (Doppelreihig). Da nur 16 Ausgänge und 8 Eingänge für die Ansteuerung der Testplatine benötigt werden, sind die restlichen Pins offen. Nicht verwendet werden Pins für z.B. 5V Eingangsspannung (für externe Spannungsversorgung), 3,3V Ausgang I²C-Bus …

Um die Mechanische Beanspruchung am Interface zu verringern wurden darauf Winkel – Steckverbindungen aufgelötet. Diese sind ebenfalls auf der I/O-Platine vorhanden. Dieses System ermöglicht ein einfaches und schnelles Anschließen verschiedenster Prüfkomponenten über Flachbandkabel.

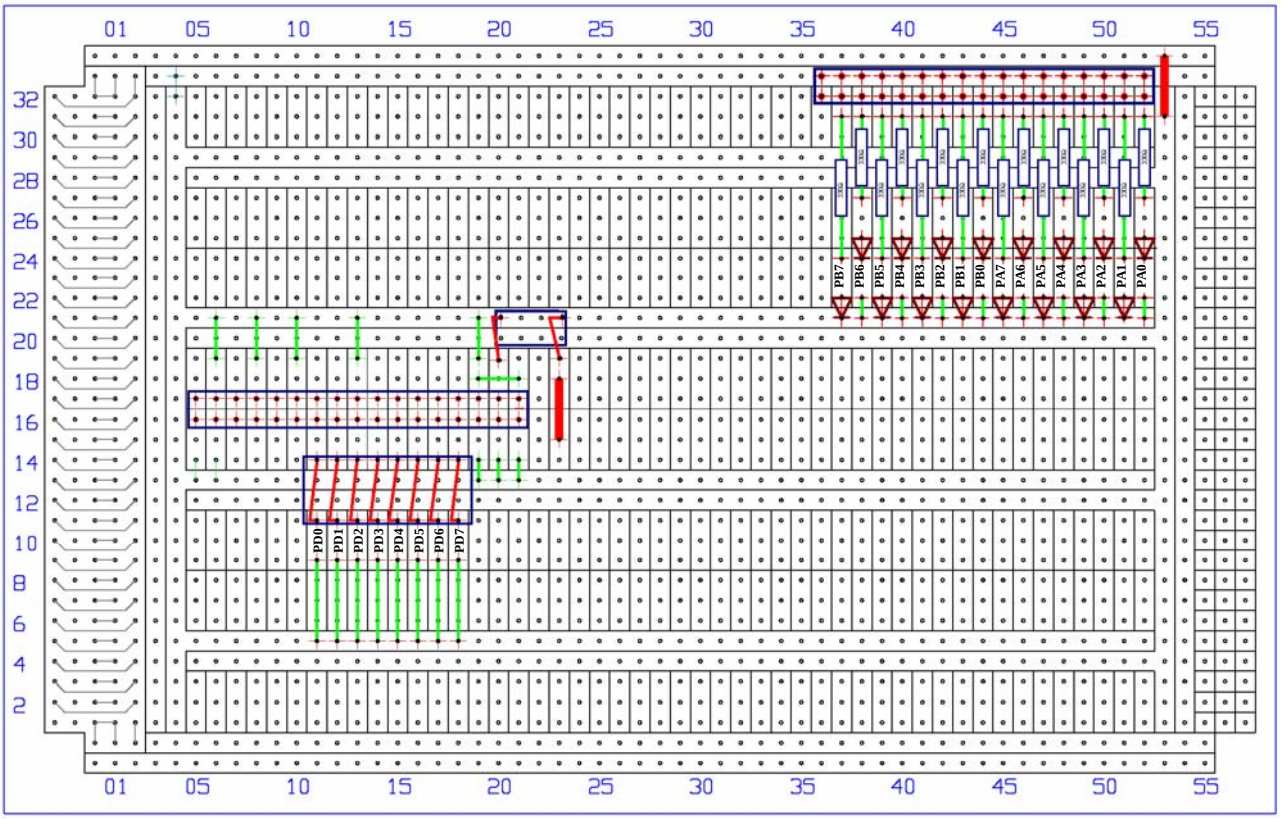
- **Stückliste:**

Laborkarte L3001	*1 Stück*
Steckverbindung SL22/124	*2 Stück*
Opto-Elektrische LED 3mm rot	*16 Stück*
Widerstand 0207 0,6W 100k	*16 Stück*
Taster 1301.9308	*1 Stück*
Schalter DS-08	*1 Stück*



(Bild I/O_Testplatine I)

Probleme:
Bei positiver Signalflanke wird der Eingang des Mikrocontrollers auf High gesetzt. Um aber ein eindeutiges Signal am Mikrocontroller-eingang zu bekommen, nachdem er auf Low gesetzt wurde, benötigt er einen Impuls von wenigen Millivolt. In der Beschaltung unserer Platine wurde dies aber nicht berücksichtigt. Nach dem Masseschluss steht dem Mikrocontroller kein weiteres Signal zur Verfügung.



(Schaltplan I/O_Testplatine I)

3.4 I/O_Testplatine II

Da die Testplatine I noch einige Fehler im Bereich der Eingangssignaleinstellung hat und weitere Änderungen umständlich und wenig sinnvoll erscheinen ist eine weiter Testplatine entwickelt worden → I/O_Testplatine II.
Die Testplatine ist jetzt auch in zwei Komponenten unterteilt, da wir die Verwendung der Flachbandkabel vermeiden wollen, um Signalprobleme zu vermeiden (Wackelkontakte, da die Floppy-Flachbandkabel nicht auf so hohe Belastungen ausgelegt sind).
Für den gleichen abstand zwischen den LED's und den Platinen werden 5mm Abstandshalter verwendet.

3.5 Funktionsweise

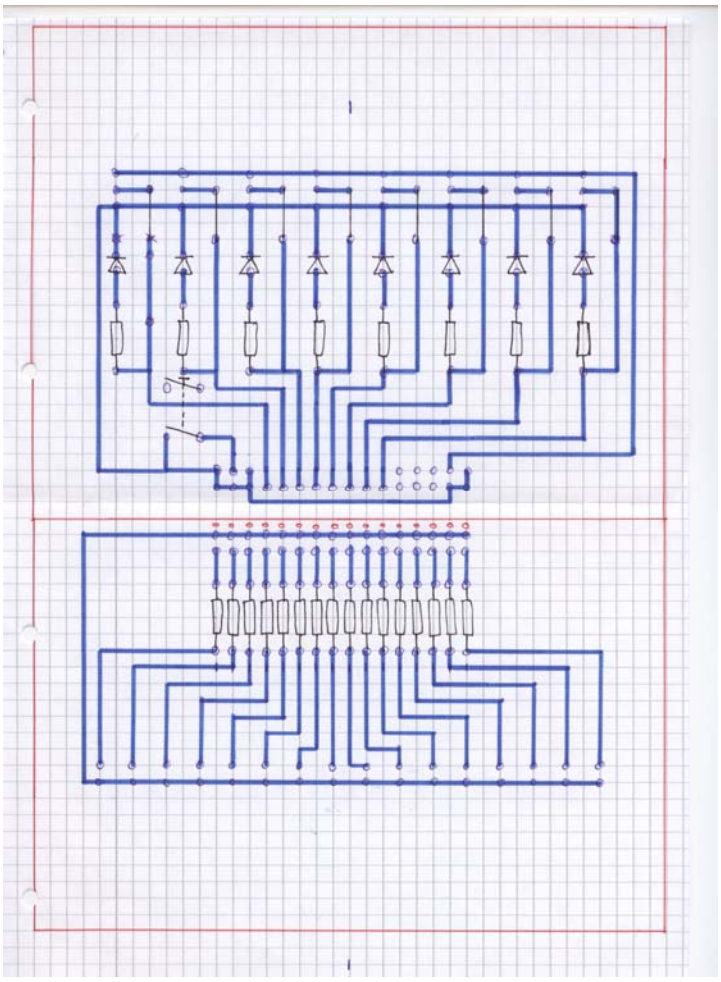
Wie Testplatine I nur mit den folgenden Unterschieden:

- die Eingänge (PD0 – PD7) werden über einen Wechselschalter zwischen Masse und +5V geschaltet
- Die Zustände der Eingänge werden mit roten LED's angezeigt.

3.6 Aufbau

- **Stückliste:**

Lochrasterplatine ..... 1 Stück
Steckverbindung SL22/124 ... 2 Stück
Opto-Elektrische LED 3mm rot ..... 24 Stück
Widerstand 330Ω ..... 24 Stück
Taster 1301.9308 ..... 1 Stück
Kippschalter ..... 8 Stück



Aus Zeitgründen konnte die I/O_Testplatine II nicht Termingerecht fertiggestellt werden!

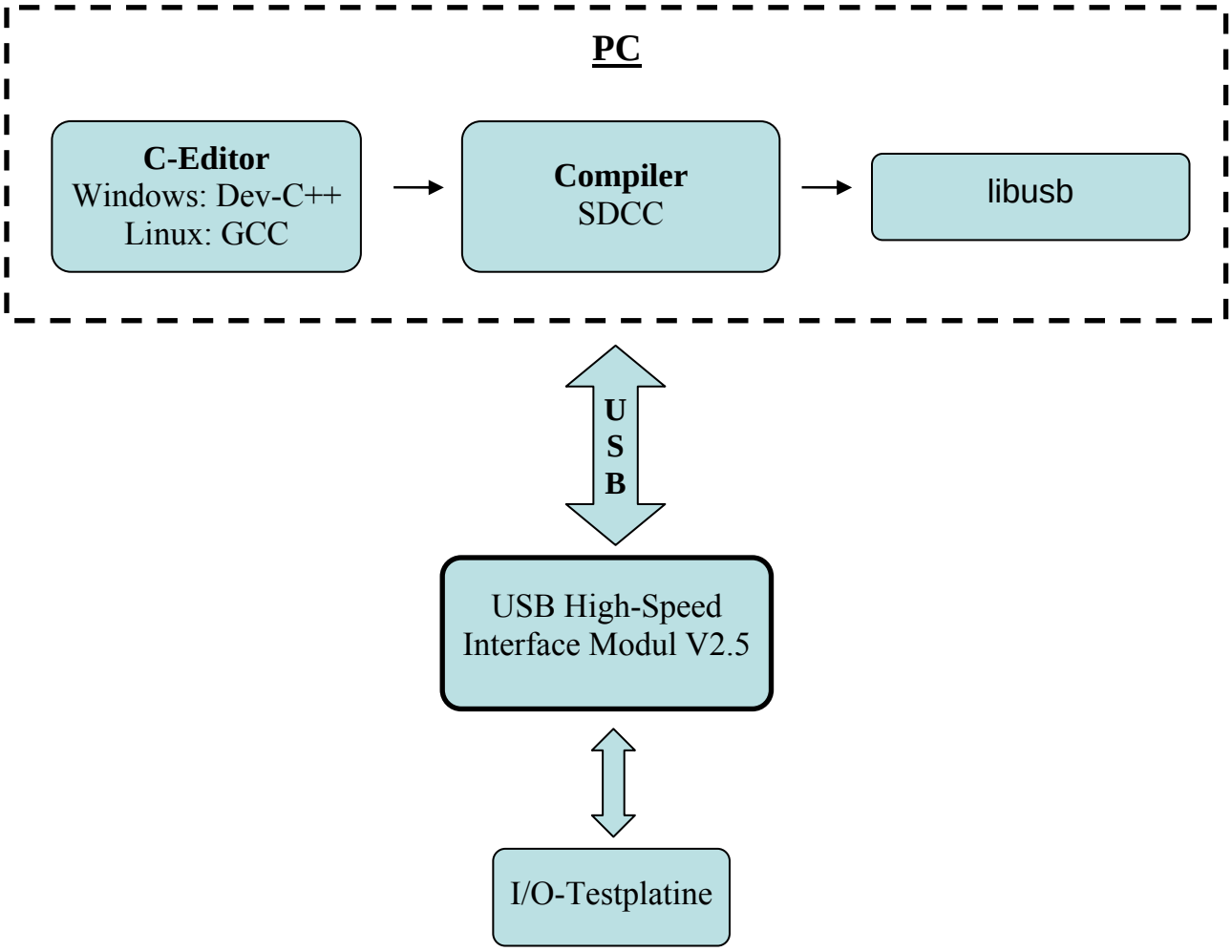
4. Software

4.1 Grundlagen

Development Software (Open Source):

- fx2_programmer
- libusb
- USB2.dll mit gleichnamiger Demo Software von Braintechnology für Windows
- EZ-USB-Treiber von Cypress sowie die Arbeitsoberfläche EZ-USB-Console für Windows
- SDCC-Kompiler für Windows und Linux
- FX-Loader für das Laden der Firmware (*.hex-files) für Linux
- DevC^{++} Kompiler (für erste Programmierversuche)

Anforderung: Zuerst muss die USB Interfacekarte vom Betriebsystem erkannt werden. Hierfür wurde die entsprechende Treibersoftware der Fa. Cypress installiert. Für den ersten Test einer Kommunikation zwischen Betriebssystem und Interfacekarte wir die USB2.DLL Demo Software verwendet. Hiermit können die entsprechenden Ports auf der Karte als Ein- oder Ausgang gesetzt werden (mit entsprechenden Signal I/0). Für die Verwendung eines eigenen „Hello World“ Programms wird ein mit Dev-C^{++} kompiliertes Programm und mit SDCC generiertes cfile.hex via EZ-USB Console auf die Karte übertragen.



4.2 Windows XP

Erste Versuche zur Inbetriebnahme des Interfaces wurden unter Windows XP vorgenommen. Als C-Editor ist der freie C-Compiler Dev-C^{++} verwendet worden. Mit dem Emulator Cygwin haben wir eine UNIX Plattform unter Windows simuliert was jedoch zu Schwierigkeiten bei der Programmierung führte da Cygwin mehr auf Software als auf Hardwareebene gerichtet ist. Alle anderen Ansätze waren nicht proporitär und würden die Kosten unnötig in die Höhe treiben.

4.3 Linux

Da unter Windows einige Komplikationen auftraten fanden alle weiteren Programmierungen unter Linux (XANDROS, FEDORA, SUSE) statt. Auf dem Laborrechner ist XANDROS installiert, SUSE und FEDORA Linux wurden an den Heimrechnern zur Entwicklung eingesetzt.

4.4 Xandros

Pakete update/ nachtragen
(Achtung! Passwort ist unverschlüsselt und sichtbar, deswegen unbedingt nach erfolgreicher Installation History löschen!)

Proxy eintragen: export http_proxy=http://username;
password@proxy.fh-karlsruhe.de:8888

Nach Paketen suchen: apt (advanced package tool)
apt-cache search paketname

Pakete installieren: apt-get install paketname

History löschen!!!: In's home Verzeichnis (cd)
rm.bash_history

4.5 fxload

Ist ein Programm, welches Firmware auf USB Geräte mit Mikrocontrollern spielen kann. FX2 unterstützt USB 2.0 und somit high-speed Übertragungsgeschwindigkeiten (480 Mbit/s).

[nähere Informationen sowie Befehle sind im Manual zu finden ***man fxload***]

4.6 fx2_programmer

Der fx2_programmer ist ein kleines Dienstprogramm welches den Zugriff auf den Halbleiterbaustein CY7C68013 von Cypress ermöglicht. Es erlaubt das Einlesen von binären sowie hex Dateien in den RAM Speicher. Auch bei der Verwaltung von Speicherinhalten und Setzen von USB Endpunkte Hilft das Programm. Das Programm ermöglicht auch den Upload einer Firmware, Setzten eines Adressbytes und Durchführen eines Benchmarks.

How to use:

fx2_programmer bus device function [parameters]

Function	Parameters	Description
dump_busses		show all available devices
dump	start len	dump RAM contents
bulk_dump	endpoint len chunk	dump data read of bulk endpoint
bulk_bench	endpoint len chunk	benchmark throughput of bulk endpoint
upload	file start len	upload binary file to RAM
set	address byte	changes values of a single byte
Program	file.ihx	programs fx2 using Intel hex format file

4.7 SDCC

Dient zur Kompilierung der C-Files (*.c). Mit Hilfe eines erstellten Makefiles wird vorgegeben welche Dateien vom SDCC-Kompieler kompiliert werden müssen und welche Codedateien erstellt werden sollen (*.hex). Zusätzlich beinhaltet das Makefile den Aufruf des GCC (**G**NU **C**ompiler **C**ollection) welches ein Front-End-Programm erstell (*.exe).

Das benutzen von verschiedenen SDCC Versionen ist sehr Fehlerträchtig. Unsere Programme wurden mit der Version 2.3.0 übersetzt. Die Versuche mit Version 2.4.0 sind gescheitert da die meisten Programme nicht funktioniert haben.

Ausführliche Informationen unter: http://sdcc.sourceforge.net/

http://gcc.gnu.org/

4.8 Midnightcommander MC

Midnight Commander ist ein DateiManager - ein Clone des Norton-Commander für Linux/Console. Für Linux-Einsteiger, die auch schon unter DOS, Windows oder OS/2 ähnliches benutzt haben, eine sehr angenehme Arbeitserleichterung. Einer seiner größten Vorteile ist seine Vielfalt: Maus-Unterstützung (GPM), eingebauter FTP-Client, Entpacken von sämtlichen Archiven und Paketen (die Programme sind natürlich vorher zu installieren) tar.gz, bzip, zip, rar, rpm, deb, ...Es mag ja sein, dass vieles per Kommandozeile schneller geht, aber mit mc ist alles einfach simpler, z.B. beim Kopieren, Löschen etc. von Daten, die sich mit Filtern nicht so einfach erfassen lassen. Besonders erfreulich für Anfänger, ist der eingebaute Editor mit Syntaxunterstützung für so ziemlich jede Sprache und einfacher Handhabung.

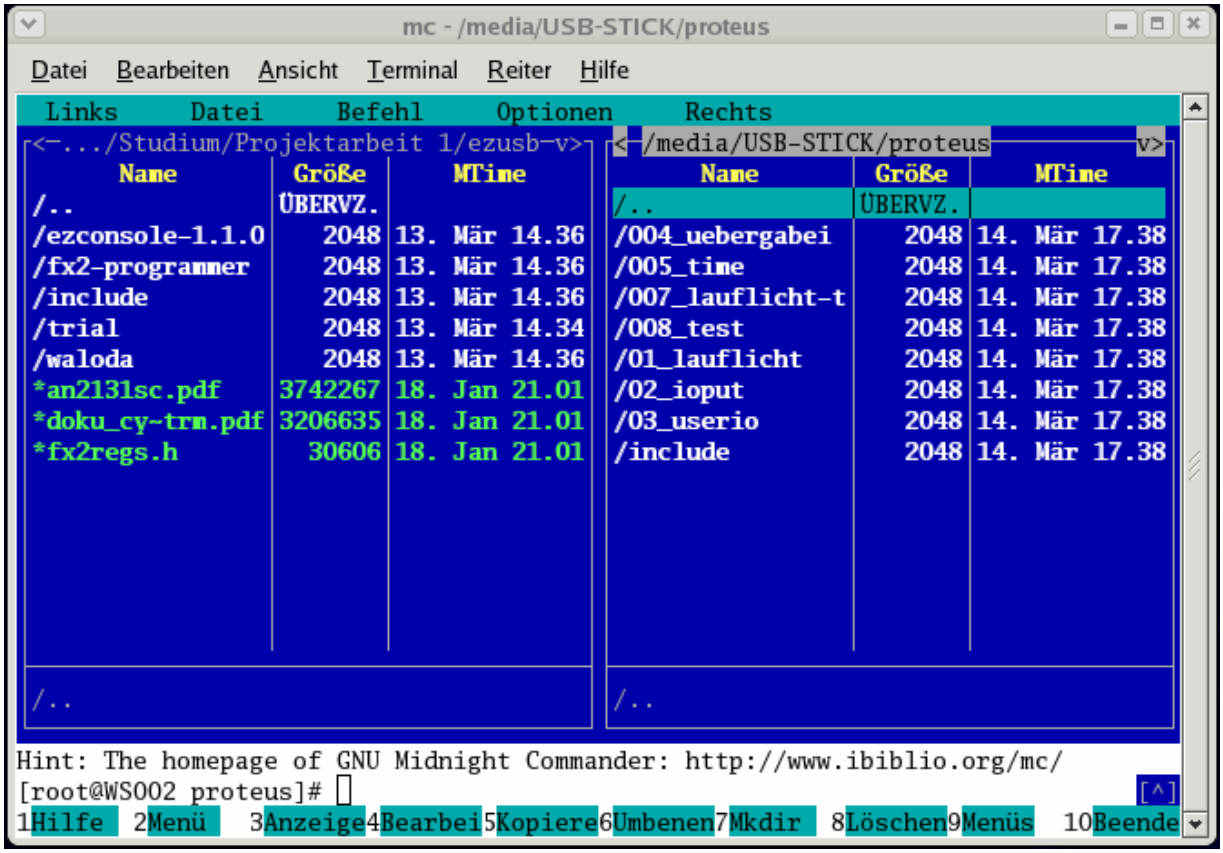
Da es eine Konsoleanwendung mit Funktionstasten ist, geht nach Eingewöhnung vieles schneller von der Hand als auf Kommandozeile oder grafischer Oberfläche. Er ist in vielen Dingen den GUI-Dateimanagern überlegen, insbesondere, was die Geschwindigkeit angeht.

FAQ's unter: http://www.ibiblio.org/mc/FAQ

Wichtige Tasten

F1 .. F10	siehe Menüleiste ab unteren Bildschirmrand
Tab	Wechseln auf anderes Panel
Strg-O	Wegklappen der Panels, so dass der normale Bildschirminhalt sichtbar wird
STRG+x c	Infomodus im anderen Fenster
STRG+x q	Quickview im anderen Fenster
STRG+s / ALT+s	Namenssuche im aktuellen Fenster

Esc-P	vorheriges Kommando
Esc-Enter	aktuellen Dateinamen auf Kommandozeile übernehmen
Esc-Tab	Kommando-Vervollständigung (ggfs. auch zweimal)
ALT-o	Fenster angleichen (anderes Fenster erhält gleichen Pfad!)
ALT-c	Wechsel in ein anzugebendes Verzeichnis
ALT+TAB	Shell-Erweiterung (nicht unter X)
ALT+?	Dateisuche
ALT+t	(togle) Wechsel zw. verschiedenen Listing-Arten
+ / -	select / unselect (pattern matching)



4.9 Linux Befehle

apropos Zeigt die Titel von man-pages zu einem gegebenen Stichwort an

ar	Archiv- und Bibliotheksverwaltung
basename	Gibt einen Dateinamen ohne Pfadangaben aus
cat	Fügt mehrere Textdateien zusammen
cd	Wechselt in das angegebene Verzeichnis
chmod	Zugriffsrechte ändern (rwx)
clear	Löscht die Konsole
cmp	Vergleicht zwei Dateien auf Übereinstimmung
compress	Komprimiert Dateien im Lempel-Ziv Verfahren
cp	Kopiert Dateien und Verzeichnisse (copy oder xcopy unter MS-DOS)
cpio	Kopiert Dateien in bzw. aus Archiven
dirname	Gibt nur den Pfad zu einer Datei aus
echo	Gibt einen Text auf der Konsole aus.
env	Gibt alle Umgebungsvariablen aus
exit	Aktuelle Session verlassen
file	Zeigt den Dateitypen einer Datei an.
find	Durchsucht den Verzeichnisbaum, ausgehend vom aktuellen Verzeichnis, nach einer Datei.
find	Umfangreiches Suchtool
gzip	Komprimiert und dekomprimiert Dateien im Lempel-Ziv Verfahren
init	Runlevel wechseln
locate	Sucht Dateien mittels eines Indexes, der durch updatedb erstellt wird
ls	Zeigt den Inhalt des aktuellen oder des angegebenen Verzeichnisses an (dir unter MS-DOS)
man	Zeigt man-pages an. (man-pages sind Seiten aus dem Linux Programmer's Manual. In diesem Manual sind so gut wie alle Programme auf Linux/Unix Basis dokumentiert)
mkdir	Legt ein neues Verzeichnis an (md unter MS-DOS)
mv	Verschiebt Dateien und Verzeichnisse (benennt auch um) (ren unter MS-DOS)
pwd	Zeigt das aktuelle Verzeichnis an
rm	Löscht Dateien und Verzeichnisse (del unter MS-DOS)
rmdir	Löscht ein Verzeichnis (rd unter MS-DOS)
strings	Extrahiert alle (lesbaren) Zeichenfolgen aus einer Datei/Eingabe
su	Neue Session mit anderem Account aufmachen (su - : root-Account).
sudo	Befehl als root ausführen.
tar	Komprimiert und dekomprimiert Archive mehrerer Dateien

unzip	Dekomprimiert zip-Archive.
updatedb	Erstellt einen Suchindex über das gesamte Dateisystem für locate
zip	Komprimiert Dateien. zip-Archive werden auch von PKZIP und WinZip (DOS/Win) verwendet

4.10 Ansteuerung der Test-Platine und dem Interface

Ablauf einer Standart Kompilierung:

Zu beachte ist, dass bei einer ersten Kompilierung die folgenden 4 Elemente nötig sind:

1. Bibliotheken z.B. ***usb.h, fx2regs.h*** etc.
2. Makefile
3. Shell-script
4. Programmdateien

Erklärung zu den jeweiligen Elemeten:

- Die Standartbibliotheken

Hierbei handelt es sich um die in ANSI-C üblichen Standartbibliotheken. Beispielsweise ruft der Befehlt # include <stdio.h> die Standartbibliothek aller gängigen Befehle in C auf. Die Bibliothek iostream.h schließt z.B. noch weitere vordefinierte Anweisungen für C^{++} ein. Um das USB-Device ansteuern zu können und den dafür vorgesehenen USB-Bus anzusprechen, muss die Bibliothek usb.h sowie für den FX2-Baustein die fx2regs.h eingebunden werden. Dabei ist besonders penibel darauf zu achten in welchem Verzeichnis sich diese Bibliotheken befinden. Denn falsche Pfadangaben führen zu keinem Kompilierungsergebnis!

- Das Makefile

Das Makefile wird benötigt um die jeweiligen Kompilierungsanweisungen an die Compiler weiterzugeben. Das geschieht wie folgt.

Beispiel der Datei ***makefile:***

```
INCLUDES=-I .     // hier wird der Ort der h-Files Festgelegt

CC=sdcc -mmcs51   // Compiler Definition in diesem Fall sdcc
GCC=gcc -pipe –g  // Compiler Definition, hier gcc (für Front-End-Programm)
FILE=test         // Dateibezeichnug die zu kompilieren ist ( z.b. test )


all: $(FILE).c $(FILE)-front.c
      $(CC) $(INCLUDES) $(FILE).c
      $(GCC) $(FILE)-front.c -lusb -o $(FILE)-front
```

Hier werden die durch die Kompilierung auszugebenden Files bestimmt. Dies geschieht in unserem Fall getrennt nach Compiler. Der Linux shell befehl lautet hierbei: ***make***.

```
clean:
      rm -f $(FILE).ihx $(FILE).rel $(FILE).rst $(FILE).lnk $(FILE).lst
$(FILE).map $(FILE).asm $(FILE).sym
```

Hier werden die Dateien angegeben, die mit dem Linux Shell befehl: ***make clean*** gelöscht werden sollen.

- Das shell-script test.sh Datei:

Es enthält im Wesentlichen die Anweisungen für den 8051µC und den fx2_programmer. Es hält den µC an und ermöglicht damit das Beschreiben des Prozessors, sprich den Upload der Firmware vom PC auf das Interface. Es ist darauf zu achten, dass die ***.ihx** Datei richtig bezeichnet ist. Details dazu, siehe Beispiel:

Beispiel der Datei ***test.sh:***

```
#!/bin/bash

make || exit

DESCR=`fx2_programmer any any dump_busses | grep UNCONFIGURED | head -n 1`
echo "Using device $DESCR"

BUS=`echo "$DESCR" | cut -f 3 -d \  `
DEVICE=`echo "$DESCR" | cut -f 5 -d \  `

#
# put 8051 into reset
#
fx2_programmer $BUS $DEVICE set 0xE600 1
#
# program 8051
#
fx2_programmer $BUS $DEVICE program test.ihx
#
# take 8051 out of reset
#
fx2_programmer $BUS $DEVICE set 0xE600 0
#
# We're good to go...
#
echo "Firmware upload done - 8051 running"
#
#execute x-front
#
```

Quellcode wird neu kompiliert.

Suche nach dem USB-BUS bzw. dem USB-Device

Microkontroller wird angehalten.

Hier ist stets der Programmname abzuändern. In diesem fall ***test.ihx***.

Microkontroller wird wieder gestartet.

Anweisung um bei der Kompilierung bzw. dem Ausführen der test.sh gleichzeitig die Linux Shell Anweisung: ***test-front*** auszuführen.

```
./test-front
```

- Die Programmdatei:

Stellt den Kern eines Kompilierungs-Projektes dar. Sie besteht zum einen aus der ***-front.c*** und zum anderen der ****.c*** Datei.

1.) Die ***-front.c*** Datei enthält den eigentlichen Quellcode. Hier werden die Standardbibliotheken eingebunden, der USB-Bus initialisiert und lokalisiert. Dann folgt die main-Funktion, welche die einzelnen Unterprogramme aufruft und den eigentlichen Funktionsteil bildet. Mit dieser Datei ist eine Shell basierende Programmierung möglich, d.h. Benutzerseitige Eingaben sind möglich (siehe Tutorials).

2.) Die ****.c*** Datei enthält grundlegende Definitionen, die den FX2-Chip initialisieren und die verschiedenen anwählbaren Ein- und Ausgänge setzen, beschreiben oder abfragen. In dieser Datei sind auch kleine Assembler-Programmierelemente enthalten, z.B. wird das „Anhalten“ des Prozessors mit dem befehl NOP definiert, um das Schreiben auf den µC zu ermöglichen. Ansonsten ist der Aufbau des Programms, dem des ****-front.c*** Files sehr ähnlich. Zuerst wird der FX2-Prozessor initialisiert, anschließend die In- und Output Ports gesetzt.
Folglich ist dies die Programmtechnisch wichtigere Datei, da hier ebenfalls in der main – Funktion Zustände am Interface abgefragt werden können. Man könnte auch hier z.B. ein Lauflicht- Programm schreiben, jedoch hat man dabei nicht die Möglichkeit über die Linux shell oder in der Sprache der Windowsuser, die gewohnte Konsolenanwendung, in den Ablauf des Programms direkt einzugreifen.

5. Diagnose / Test

5.1 Programm- Beispiele

Im folgenden Teil werden einige Tutorials als Beispiel dargestellt. Es handelt sich hierbei um den Ordner Proteus-Tutorials. Dieser ist auch auf der Daten-CD zu finden. Begonnen wird mit dem Beispiel Programm 01_lauflicht, das die grundlegende Arbeitsweise verdeutlichen soll. Anschließend folgen der Reihe nach 02_ioput, 03_uebergabei usw.

Einführende Erklärung:

Wie in 4.10 erwähnt, benötigt jedes Programm das geschrieben wird 4 Kompilierungselemente. Diese müssen sich alle im selben Ordner befinden. Es ist besonders darauf zu achten die richtigen Pfadangaben zu jeglichen Bibliotheken sicherzustellen. Die Bezeichnung der Ordner für das Programm, in unseren Beispielen 01_lauflicht, sind beliebig zu wählen. Aber die einzelnen Dateien z.B. 01_lauflicht.c zu nennen, ist nicht empfehlenswert, da man Probleme beim Kompilieren bekommt. Aus diesem Grund haben wir die Dateien ohne dem 01_ etc. Kürzel umbenannt.

Tutorials:

5.1.1 lauflicht

Das Programm steuert der reihe nach die Ports B7-B0 und dann A0-A7 an. Die auf der I/O_Testplatine I befindlichen LED's werden dadurch geschaltet und ein Lauflicht wird dadurch simuliert.
Beispiel zur Datei ***lauflicht.c***

Dieses Programm dient als Einführung in den Programmaufbau und Ablauf.

```
#define ALLOCATE_EXTERN
#include <fx2regs.h>

#define BLOCKSIZE 64

#define NOP   _asm \
           nop; \
           _endasm;
#define PORTOUTA     IOA
#define PORTOUTB     IOB
#define PORTOUTCFGA  OEA
#define PORTOUTCFGB  OEB
#define PORTIN       IOD
#define PORTINCFG    OED
#define PORTIND      PD0
```

```
void init_fx2(void)
{
    CPUCS = 0x10;             // 48MHz CPU-Clock
    EP1OUTBC = 0x01;          // Writing to EP1OUTBC rearms for an out Transfer
    NOP;
    PORTOUTCFGA = 0xff;       // Ausgabeport auf Ausgabe stellen
    NOP;
    PORTOUTCFGB = 0xff;
    NOP;
    PORTOUTA = 0x00;          // Auf Null setzen...
    NOP;
    PORTOUTB = 0x00;
    NOP;
    PORTINCFG = 0xff;         // Eingabeport auf Ausgabe
    NOP;
    PORTIN = 0x00;            // Einmal auf Null stellen
    NOP;
    PORTINCFG = 0x00;         // Jetzt Eingabeport auf Eingabe einstellen
}
void main(void)
{
    int i,temp;
    long x;

    init_fx2();               // Unterfunktionsaufruf

    while(1)                  // Endlosschleife
    {
    for(i=0; i<16; i++)       // Zaehlkriterium
    {
      temp = i;

    for(x=0; x<35000; x++)    // Pausenkriterium min. 1 - max. 10000 [t]
    {
    switch(temp)
    {
    case 0: PORTOUTB = 128,   PORTOUTA = 0x00;  break;
    case 1: PORTOUTB = 64;    break;
    case 2: PORTOUTB = 32;    break;
    case 3: PORTOUTB = 16;    break;
    case 4: PORTOUTB = 8;     break;
    case 5: PORTOUTB = 4;     break;
    case 6: PORTOUTB = 2;     break;
    case 7: PORTOUTB = 1;     break;
    case 8: PORTOUTA = 1,     PORTOUTB = 0x00;  break;
    case 9: PORTOUTA = 2;     break;
    case 10:PORTOUTA = 4;     break;
    case 11:PORTOUTA = 8;     break;
    case 12:PORTOUTA = 16;    break;
    case 13:PORTOUTA = 32;    break;
    case 14:PORTOUTA = 64;    break;
    case 15:PORTOUTA = 128;   break;

    default: PORTOUTA = 0xff,PORTOUTB = 0xff; //Soll auftretende Fehler anzeigen
    }
```

```
    }
    }
    }
}
```

5.1.2 ioput

Dieses Programm gibt eins zu eins die Eingangssignale von Port D an die Ausgangsport A weiter.

Beispiel zur Datei ioput.c

```
#define ALLOCATE_EXTERN
#include <fx2regs.h>

#define BLOCKSIZE 64

#define NOP   _asm \
           nop; \
           _endasm;
#define PORTOUTA     IOA
#define PORTOUTB     IOB
#define PORTOUTCFGA  OEA
#define PORTOUTCFGB  OEB
#define PORTIN       IOD
#define PORTINCFG    OED
#define PORTIND      PD0


void init_fx2(void)
{
    CPUCS = 0x10;             // 48MHz CPU-Clock
    EP1OUTBC = 0x01;          // Writing to EP1OUTBC rearms for an out Transfer
    NOP;
    PORTOUTCFGA = 0xff;       // Ausgabeport auf Ausgabe stellen
    NOP;
    PORTOUTCFGB = 0xff;
    NOP;
    PORTOUTA = 0x00;          // Auf Null setzen...
    NOP;
    PORTOUTB = 0x00;
    NOP;
    PORTINCFG = 0x00;         // Eingabeport auf Ausgabe
    NOP;
    PORTIN = 0x00;            // Einmal auf Null stellen
    NOP;
    PORTINCFG = 0x00;         // Jetzt Eingabeport auf Eingabe einstellen
}

void main(void)
{
    init_fx2();
```

```
    while(1)                    //Endlosschleife
    {
    PORTOUTA = PORTIN;          //Eingänge auf Ausgänge gesetzt
    }
}
```

5.1.3 userio

Dieses Programm gibt je nach Eingabe des Benutzers die entsprechenden Bitzustände auf der I/O Testplatine aus. Hier werden unter anderem die userio.c und userio-front.c Dateien benötigt.

Die userio.c Datei: Der Programmcode für das Initialisieren des Mikrocontrollers.

*Beispiel zur Datei **userio.c***

```
/*********************************************************************
*                                                                    *
*  Firmware fuer das lesen und schreiben der ports                   *
*  am FX2 device(Cypress FX2 controller - In unserem                 *
*  Fall wird das USB-Interface Version 2.5 von                       *
*  Braintechnology verwendet) (56-Pin package)                       *
*                                                                    *
*  IOA, IOB, IOD sind drei ports, die auf der 56-Pin                 *
*  package Version auf dem FX2 controller                            *
*  erreichbar sind.                                                  *
*  Diese ports(Speicheradressen) koennen je nachdem,                 *
*  wie sie adressiert worden sind gelesen bzw.                       *
*  beschrieben werden.                                               *
*  OEA, OEB, OED sind konfigurationsregister fuer                    *
*  Port A, B und D. Wenn ein Bit in einem dieser                     *
*  Register gesetzt wird, erlaubt die Firmware die                   *
*  dazugeoerigen Bits am zugehoerigen Port zu setzen.                *
*  Im Fall eines ungesetzten Bits uebernimmt der Port                *
*  alle "high" signale als eine "1". Bevor man die                   *
*  high/low Portzustaende auslesen kann, muss man                    *
*  vorher alle bits auf "0" setzen.                                  *
*                                                                    *
*  Zur Firmware:                                                     *
*  init_fx2:                                                         *
*     CPU auf hoechste waehlbare Frequenz setzen.                    *
*     Initialisierung des Endpoint 1 Buffer um ihn                   *
*     beschreiben zu koennen                                         *
*     Rueck/setzen der I/O-ports.                                    *
*  readioports:                                                      *
*     Auslesen der Variablen von IOA, IOB and IOD and                *
*     speichert diese in EP1INBUF (1-3).                             *
*  setioports:                                                       *
*     Setzt die I/O-ports auf die Werte des EP1OUTBUF                *
*     (1-3). Danach wird readioports aufgerufen um                   *
*     zu ueberpruefen ob die Werte auch an den Ports                 *
*     anliegen.                                                      *
```

```
*  getioports:                                                  *
*     Setzt alle I/O-ports zurueckum fuer die                   *
*     Ausgelesenen Werte bereit zu sein.                        *
*     Das Auslesen ist durch Aufruf der readioports             *
*     gegeben.                                                  *
*  main:                                                        *
*     Unterfunktionsaufruf: init_fx2,                           *
*     im Anschluss folgt die Endlosschleife die                 *
*     die in- und output buffer auf vorhandene Daten            *
*     ueberprueft. Die firmware entscheidet je nach             *
*     einstellung des ertsen byte im EP1OUTBUF (0),             *
*     ob die Ports gelesen, also gesetzt oder                   *
*     beschrieben werden.                                       *
*                                                               *
***************************************************************/

#define ALLOCATE_EXTERN       /* Compiler anweisung */
#include <fx2regs.h>          /* Beinhaltet alle definitionen der Register und
                              Speicher */
                              /* fuer den Zugriff auf die FX2 Hardware */
#define BLOCKSIZE 64
                              // Rechenprozess fuer einen Augenblick anhalten.
#define NOP   _asm \
            nop; \
            _endasm;

void init_fx2(void)
{
    CPUCS = 0x10;             // 48MHz CPU-Clock
    EP1INCS = EP1INCS & (0xff - bmEPSTALL);     /* Ruecksetzen der STALL bits in
                                                beide Richtungen.*/
    EP1OUTCS = EP1OUTCS & (0xff - bmEPSTALL);  //
    EP1OUTBC = 0x40;          // Schreiben auf EP1OUTBC um den out-Transfer neu
                              zu starten
    OEA = OEB = OED = 0xff;   // Port configuration fuer output (Jedes Bit wird
                              auf 1 gesetzt)
    IOA = IOB = IOD = 0xff;   // Alle Bits auf 0 setzen(Erscheinen als low
                              signal)
}

void readioports(void)
{
    EP1INBUF[1] = IOA;        // Auslesen der Werte des jeweiligen Ports
    EP1INBUF[2] = IOB;        // Werte werden in EP1INBUF 1 bis 3 gespeichert
    EP1INBUF[3] = IOD;
    EP1INCS = EP1INCS & (0xff - bmEPSTALL);     // Die Stallbits zur Sicherheit
                                                nochmals ruecksetzen
    EP1OUTCS = EP1OUTCS & (0xff - bmEPSTALL);
    EP1INBC = 4;        // endpoint 1 neu starten fuer in transfer von 4 bytes
    EP1OUTBC = 64;            // endpoint 1 for out transfers neu starten
}

void setioports(void)
{
    OEA = OEB = OED = 0xff;   // Alle bits an jedem Port auf schreiben setzen
    IOA = EP1OUTBUF[1];       // Werte via USB ausgeben
```

```
    IOB = EP1OUTBUF[2];       // (Bytes 1-3 in EP1OUTBUF, byte 0 ist das
                              Anweisungs-byte)
    IOD = EP1OUTBUF[3];       //
    readioports();            // Auslesen der I/O-port - Werte, um die Operation
                              zu ueberpruefen
    EP1INCS = EP1INCS & (0xff - bmEPSTALL);     // Sicherheitsabfrage um
                                                auftetende Fehler zu finden, die stall
    EP1OUTCS = EP1OUTCS & (0xff - bmEPSTALL);   // bits werden wieder
                                                zurueckgesetzt
    EP1INBC = 0x04;           // Neustart von endpoint 1 fuer den in transfer
                              von 4 bytes
    EP1OUTBC = 0x40;          // Neustart von endpoint 1 fuer den out transfer
}

void getioports(void)
{
    OEA = OEB = OED = 0xff;   // Setze I/O-ports auf schreib-zugriff
    IOA = IOB = IOD = 0x00;   // Setze alle bits auf "0"
    OEA = OEB = OED = 0x00;   // Setze all I/O-ports auf lese-zugriff
    readioports();            // lesen...
}

void main(void)
{
    init_fx2();               // Unterprogrammaufruf
    while (1)                 // Endlosschleife...
    {
      if (!(EP1OUTCS & bmEPBUSY))   // Pruefe auf vorhandene Daten
      {
          if (!(EP1INCS & bmEPBUSY))
            // Pruefe ob man die Werte der Ports in EP1INBUF speichern kann
            // Anschliessend die Nutzer-Anweisung pruefen
          {
            if (EP1OUTBUF[0] == 0)
                getioports();
            if (EP1OUTBUF[0] == 1)
                setioports();
            if (EP1OUTBUF[0] == 2)
                readioports();
          }
      }
    }
}
```

Die userio-front.c Datei: Programmcode für die Eingabe und Anzeige auf der Shell durch den Benutzer:

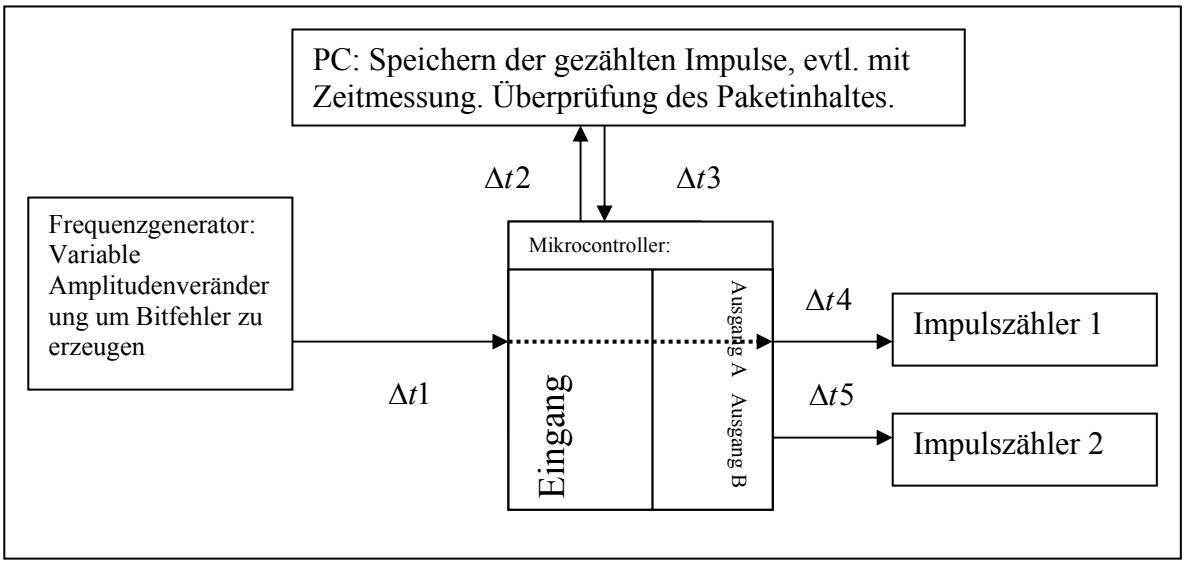
```
/*************************************************************
*                                                            *
*  Benutzerprogramm zur Initaialisierung der                 *
*   uebergabei.c firmware fuer das FX2 USB device.           *
*                                                            *
*                                                            *
*  Die Funktion des  Programms:                              *
*  Dieses Programm setzt alle outputports bzw. liest         *
*  diese aus. Um dies auszufuehren werden 3 bytes            *
*  benoetigt. Diese bytes werden gesendet bzw. empfangen     *
*                                                            *
*  Es wird ein endpoint1 benutzt, in dem die                 *
*  Ergebnisse der buffer Felder die gleiche groesse          *
*  haben. In dieser Datei uebergabei-front.c ist ein         *
*  beispiel gegeben, das den  Benutzer auffortert eine       *
*  Eingabe zu machen. Diese Eingabe wird dann auf das        *
*  FX2 device geschrieben.                                   *
*                                                            *
*  Hier wird der Anwender nach einer Zahl gefragt            *
*  die dann auf die Ausgangs-Ports geschrieben werden.       *
*  Das Interface kann auch zum lesen und schreiben           *
*  erweitert werden.                                         *
*                                                            *
*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <usb.h>

#define MAX_CHUNKSIZE  64

struct usb_device *current_device;
usb_dev_handle *current_handle;

struct usb_device *locate_usb_dev(void)   // die Funktion gibt einen pointer an
                                          das FX2 USB-device
{
    struct usb_bus *bus_search;     // Variablen definition f|r einen Bus und
                                    zwei Devices
    struct usb_device *device_search;     // Das erste FX2 device, das gefunden
                                          wird, wird gespeichert
    struct usb_device *device_found;
    device_found = NULL;      // NULL-Pointer wird gesetzt, damit kann man
                              spaeter ueberpruefen ob ein Device gefunden
                        // wurde oder nicht. NULL-Pointer zeigt auf "a".
    bus_search = usb_busses;  // "p" ist der pointer der auf den ersten Bus auf
                              der bus-liste
                       // (Initializes by usb_find_busses() ) zeigt.
    printf("Dump of USB subsystem:\n");
    while (bus_search != NULL)      // auf dem Bus nach devices suchen...
    {
      device_search = bus_search->devices;      // "q" zeigt auf das jeweilige
                                                Device des jeweiligen Buses.
      while (device_search != NULL)             // Suchschleife durchfuehren ...
```

```
      {
      printf("bus %s device %s vendor id=0x%04x product id=0x%04x %s\n",
             bus_search->dirname, device_search->filename,
             device_search->descriptor.idVendor,
             device_search->descriptor.idProduct,
             (device_search->descriptor.idVendor == 0x4b4)
             && (device_search->descriptor.idProduct == 0x8613) ?
             (device_found == NULL ? "(FX2 Device - USED)" : "(FX2 Device -
Unused)") : "");
            // Informationen ueber das Device werden angezeigt: ob FX2 gefunden
            oder genutzt wird.
          if ((device_search->descriptor.idVendor == 0x4b4)
            && (device_search->descriptor.idProduct == 0x8613)
            && (device_found == NULL))
            device_found = device_search;
            // In diesem Fall ist "a" noch NULL-Pointer und FX2 Device wurde
            gefunden.
          device_search = device_search->next;  // Naechstes device in der
                                                devices-liste
      }
      bus_search = bus_search->next;      // Naechster bus in der bus-liste
    }
    fflush(stdout);           // stdout wird geschrieben
    return device_found;      // Funktionsrueckgabe
}

void write_bulkdata(int len, unsigned char *buffer,
                usb_dev_handle * writeto)
{
    int a;
    if (usb_claim_interface(writeto, 0) < 0)
    {
      fprintf(stderr, "Could not claim interface 0: %s\n", usb_strerror());
      return;
    }
    usb_set_altinterface(writeto, 1);
    a = usb_bulk_write(writeto, 0x01, buffer, len, 1000);
    if (a < 0)
    {
      fprintf(stderr, "Request for bulk write of %d bytes failed: %s\n",
            len, usb_strerror());
      usb_release_interface(writeto, 0);
      return;
    }
    fflush(stdout);
    usb_release_interface(writeto, 0);
}
void read_bulkdata(int len, unsigned char *buffer, usb_dev_handle * readof)
{
    int i;
    int a;
    if (usb_claim_interface(readof, 0) < 0)
    {
      fprintf(stderr, "Could not claim interface 0: %s\n", usb_strerror());
      return;
    }
```

```
    usb_set_altinterface(readof, 1);
    a = usb_bulk_read(readof, 0x81, buffer, len, 1000);
    if (a < 0)
    {
      fprintf(stderr, "Request for bulk read of %d bytes failed: %s\n", len,
usb_strerror());
      usb_release_interface(readof, 0);
      return;
    }
    fflush(stdout);
    usb_release_interface(readof, 0);
}

int main(int argc, char *argv[])
{
    unsigned long output_number;    // Long Variable um Input zu lesen
    unsigned char iobuffer[MAX_CHUNKSIZE];      // Buffer fuer USB-I/O-
                                                Operationen
    usb_init();               // Funktionsaufruf der libusb(Initaialisierung)
    usb_find_busses();        // Funktionsaufruf des usb_busses
    usb_find_devices();       // Funktionsaufruf der device-lists
    current_device = locate_usb_dev();
    if (current_device == NULL)     // In diesem Fall wird kein Device gefunden
                                    - Gibt einen Fehler aus
    {
      fprintf(stderr, "Cannot find FX2 Device on any Bus\n");
      return -1;
    }
    current_handle = usb_open(current_device);    // Zugriff(handle) auf das
                                                     gesuchte device
    printf("Bitte geben sie eine Zahl ein (Maximal 65535(2^16 -1): ");
    scanf("%d", &output_number);    // Eingabe der Zahl die an den USB-Device-
                                    I/O-Ports ausgegeben wird
    iobuffer[0] = (char) 0x01;      // Anweisung an das USB-Device, die Zahl zu
                                    schreiben
    iobuffer[1] = (char) output_number % 256;  // Aufteilen der output_number in
                                                3 byte-variablen...
    iobuffer[2] = (char) (output_number >> 8) % 256;
    iobuffer[3] = (char) (output_number >> 16) % 256;
    write_bulkdata(4, &iobuffer[0], current_handle);    // Speicher auf das USB-
                                                           device schreiben
    read_bulkdata(4, &iobuffer[0], current_handle);    // Lese vom USB-device
                                                          nach "c"
    printf("Rueckgabe IOA: %d, IOB: %d IOD: %d\n", ((long) iobuffer[1]),
         ((long) iobuffer[2]), ((long) iobuffer[3]));    // Ausgabe der
                                                         Variablen fuer jeden Port
    usb_close(current_handle);      // current handle beenden
    return 0;
}
```

5.1.4 test

Dies ist der stand unserer Entwicklung. Wir versuchen eine Latenzzeitmessung durchzuführen. Dies soll folgendermaßen geschehen:
Auf einen Eingang wird ein Rechtecksignal mittels Frequenzgenerator(1) gelegt, dessen Frequenz erhöht wird. Diese Impulse werden am Ausgang gemessen und durch den Impulszähler(1) gezählt. Zur gleichen Zeit sollen kleine "USB-Pakete" gepackt werden, die die Zahl der Impulse beinhalten. Diese werden dann am Bildschirm ausgegeben oder/und in einer Variablen abgespeichert. Um zu überprüfen ob das Paket auch alle Daten ohne Fehler übertagen hat oder sonstige Zählfehler enthält, sollte dies überprüft werden. Ein rampenähnlicher Zählanstieg ist z.B. gegeben und Änderungen können so leicht festgestellt werden.
Anschließend wird das Paket wieder an das Interface geschickt und über einen zweiten Ausgang ebenfalls an einen Impulszähler(2) geschickt. Dort müssten dann, wenn keine Verluste aufgetreten sind, die gleichen Impulse gezählt werden wie am Eingang, also mit der Impulsrate des Frequenzgenerators(1) übereinstimmen.



Letztendlich können dann alle Δt 's erfasst werden und ergeben im Mittel die Latenzzeit.

$Latenz = \frac{(\Delta t5 - \Delta t3) + (\Delta t2 - \Delta t1)}{2}$ Über $\Delta t1$ und $\Delta t4$ lässt sich ebenfalls noch den Jitter bestimmen.

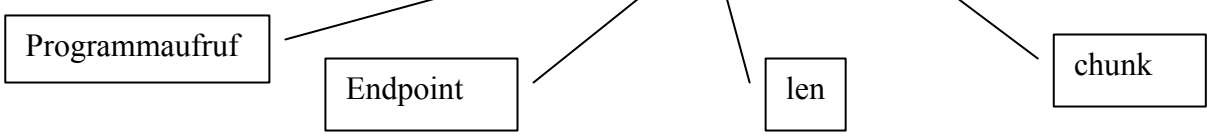
Bei der Informationsrecherche sind folgende Fehler, bzw. Problematiken aufgetreten:

- Die Verbindung zwischen der *.sh Datei und dem FX2-Programmer vollständig zu entschlüsseln.
- Die Übergabe der Parameter in der *.sh Datei und deren Auswirkungen auf den FX2-Programmer und dessen Variablen.
- Jens Dopke hat nach derzeitigem kenntnisstand die Elemente aus dem FX2-Programmer übernommen und in die *-front.c Datei eingefügt.
- Programmierer Waloda hat viele Trials und Programmcode bereitgestellt, nur ist leider wenig bis nichts dokumentiert.

- Der Zusammenhang zwischen Buffer und I/O-Signalübergabe ist noch nicht geklärt.
- Eine Verbindung zwischen eingelesenen Signalen am Interface und dem PC-Speicher herzustellen.

Zusätzliches zum fx2_programmer (Kapitel 4.6)
Hier ein Auszug der fx2_programmer.c Datei. Mit Hilfe der *.sh Datei können verschiedene Parameter verändert werden, die sich je nach einstellung auf die fx2_programmer Datei auswirken. Siehe Beispiel Program_and_start.sh Datei.

```
#!/bin/bash

make || exit

DESCR=`fx2_programmer any any dump_busses | grep UNCONFIGURED | head -n 1`
echo "Using device $DESCR"

BUS=`echo "$DESCR" | cut -f 3 -d \  `
DEVICE=`echo "$DESCR" | cut -f 5 -d \  `

#
# put 8051 into reset
#
fx2_programmer $BUS $DEVICE set 0xE600 1
#
# program 8051
#
fx2_programmer $BUS $DEVICE program ex2.ihx
#
# take 8051 out of reset
#
fx2_programmer $BUS $DEVICE set 0xE600 0
#
# dump results of the computation (which is finished by now..)
#
fx2_programmer $BUS $DEVICE bulk_dump 0x88 640 64
fx2_programmer $BUS $DEVICE bulk_bench 0x88 13107200 131072
```



Die fx2_programmer.c Datei:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
```

```
#include <usb.h>

void *do_alloc(long a, long b)
{
void *p;
if(a<1)a=1;
if(b<1)b=1;
p=calloc(a,b);
while(p==NULL){
      fprintf(stderr,"Failed to allocate %ld chunks of %ld bytes each (%ld bytes
total)\n", a,b,a*b);
      sleep(1);
      p=calloc(a,b);
      }
return p;
}

int atoz(char *s)
{
int a;
if(!strncasecmp("0x", s, 2)){
      sscanf(s, "%x", &a);
      return a;
      }
return atoi(s);
}

void dump_busses(void)
{
struct usb_bus *p;
struct usb_device *q;
p=usb_busses;
printf("Dump of USB subsystem:\n");
while(p!=NULL){
      q=p->devices;
      while(q!=NULL){
            printf(" bus %s device %s vendor id=0x%04x product id=0x%04x %s\n",
                  p->dirname, q->filename, q->descriptor.idVendor, q-
>descriptor.idProduct,
                  (q->descriptor.idVendor==0x4b4) && (q-
>descriptor.idProduct==0x8613)?
                  "(UNCONFIGURED FX2)":"");
            q=q->next;
            }
      p=p->next;
      }
fflush(stdout);
}

struct usb_device *find_device(char *busname, char *devicename)
{
struct usb_bus *p;
struct usb_device *q;
p=usb_busses;
while(p!=NULL){
      q=p->devices;
```

```
      if(strcmp(p->dirname, busname)){
            p=p->next;
            continue;
            }
      while(q!=NULL){
            if(!strcmp(q->filename, devicename))return q;
            q=q->next;
            }
      p=p->next;
      }
return NULL;
}

struct usb_device *current_device;
usb_dev_handle *current_handle;

void dump_ram(int start, int len)
{
unsigned char buffer[64];
int i;
int tlen;
int quanta=16;
int a;
for(i=start;i<start+len;i+=quanta){
      tlen=len+start-i;
      if(tlen>quanta)tlen=quanta;
      a=usb_control_msg(current_handle, 0xc0, 0xa0, i, 0, buffer, tlen, 1000);
      if(a<0){
            fprintf(stderr,"Request to download ram contents failed: %s\n",
usb_strerror());
            return;
            }
      printf("0x%04x:", i);
      for(a=0;a<tlen;a++)printf(" %02x", buffer[a]);
      printf("\n");
      }

fflush(stdout);
}

#define MAX_CHUNKSIZE  128*1024

void dump_bulkdata(int endpoint, int len, int chunk)
{
unsigned char buffer[MAX_CHUNKSIZE];
int i;
int tlen;
int a;
if(chunk>MAX_CHUNKSIZE){
      fprintf(stderr,"Unsupported chunk value: %d, should be less or equal to
%d\n", chunk, MAX_CHUNKSIZE);
      return;
      }
if(usb_claim_interface(current_handle, 0)<0){
      fprintf(stderr,"Could not claim interface 0: %s\n", usb_strerror());
      return;
```

```
      }
usb_set_altinterface(current_handle, 1);
for(i=0;i<len;i+=chunk){
      tlen=len-i;
      if(tlen>chunk)tlen=chunk;
      a=usb_bulk_read(current_handle, endpoint, buffer, tlen, 1000);
      if(a<0){
            fprintf(stderr,"Request for bulk read of %d bytes failed: %s\n",
tlen, usb_strerror());
            usb_release_interface(current_handle, 0);
            return;
            }
      printf("0x%04x:", i);
      for(a=0;a<tlen;a++)printf(" %02x", buffer[a]);
      printf("\n");
      }
fflush(stdout);
usb_release_interface(current_handle, 0);
}


void bench_bulk(int endpoint, int len, int chunk)
{
unsigned char buffer[MAX_CHUNKSIZE];
int i;
int tlen;
int a;
struct timeval tv1,tv2;
long long usec;
if(chunk>MAX_CHUNKSIZE){
      fprintf(stderr,"Unsupported chunk value: %d, should be less or equal to
%d\n", chunk, MAX_CHUNKSIZE);
      return;
      }
if(usb_claim_interface(current_handle, 0)<0){
      fprintf(stderr,"Could not claim interface 0: %s\n", usb_strerror());
      return;
      }
usb_set_altinterface(current_handle, 1);
gettimeofday(&tv1, NULL);
for(i=0;i<len;i+=chunk){
      tlen=len-i;
      if(tlen>chunk)tlen=chunk;
      a=usb_bulk_read(current_handle, endpoint, buffer, tlen, 1000);
      if(a<0){
            fprintf(stderr,"Request for bulk read failed: %s\n",
usb_strerror());
            usb_release_interface(current_handle, 0);
            return;
            }
      }
gettimeofday(&tv2, NULL);
usec=tv2.tv_sec*1000000+tv2.tv_usec;
usec-=tv1.tv_sec*1000000+tv1.tv_usec;
printf("%d bytes in %ld Sekunden und %ld Mikrosekunden gelesen. Bei einer Rate
von %d bytes/Sekunde\n",
```

```
      len, (long)usec/1000000, (long)usec % 1000000, (long)(((long
long)len*1000000)/usec));
fflush(stdout);
usb_release_interface(current_handle, 0);
}

void upload_ram(unsigned char *buf, int start, int len)
{
int i;
int tlen;
int quanta=16;
int a;
for(i=start;i<start+len;i+=quanta){
      tlen=len+start-i;
      if(tlen>quanta)tlen=quanta;
      a=usb_control_msg(current_handle, 0x40, 0xa0, i, 0, buf+(i-start), tlen,
1000);
      if(a<0){
            fprintf(stderr,"Request to upload ram contents failed: %s\n",
usb_strerror());
            return;
            }
      }

}

void upload_file(char *filename, int start, int len)
{
char *buf;
FILE *f;
buf=do_alloc(len, sizeof(char));
f=fopen(filename, "r");
if(f==NULL){
      fprintf(stderr,"Cannot open file \"%s\" for reading:");
      perror("");
      return;
      }
fread(buf, 1, len, f);
upload_ram(buf, start, len);
fclose(f);
}

void program_fx2(char *filename)
{
FILE *f;
unsigned char s[1024];
int length;
int addr;
int type;
unsigned char data[256];
unsigned char checksum,a;
unsigned int b;
int i;
f=fopen(filename, "r");
if(f==NULL){
      fprintf(stderr,"Cannot open file \"%s\" for reading:");
```

```
      perror("");
      return;
      }
printf("Using file \"%s\"\n", filename);
while(!feof(f)){
      fgets(s, 1024, f); /* we should not use more than 263 bytes normally */
      if(s[0]!=':'){
            fprintf(stderr,"%s: invalid string: \"%s\"\n", filename, s);
            continue;
            }
      sscanf(s+1, "%02x", &length);
      sscanf(s+3, "%04x", &addr);
      sscanf(s+7, "%02x", &type);
      if(type==0){
            printf("Programming %3d byte%s starting at 0x%04x", length,
length==1?" ":"s", addr);
            a=length+(addr &0xff)+(addr>>8)+type;
            for(i=0;i<length;i++){
                  sscanf(s+9+i*2,"%02x", &b);
                  data[i]=b;
                  a=a+data[i];
                  }
            sscanf(s+9+length*2,"%02x", &b);
            checksum=b;
            if(((a+checksum)&0xff)!=0x00){
                  printf("  ** Checksum failed: got 0x%02x versus 0x%02x\n", (-
a)&0xff, checksum);
                  continue;
                  } else {
                  printf(", checksum ok\n");
                  }
            upload_ram(data, addr, length);
            } else
      if(type==0x01){
            printf("End of file\n");
            fclose(f);
            return;
            } else
      if(type==0x02){
            printf("Extended address: whatever I do with it ?\n");
            continue;
            }
      }
fclose(f);
}

void show_help(void)
{
printf( "\nfx2_programmer  bus device function [parameters]\n"
      "\n"
      "       Function    Parameters            Description\n"
      "       dump_busses                       show all available devices\n"
      "       dump        start len             dump RAM contents\n"
      "       bulk_dump   endpoint len chunk    dump data read of bulk
endpoint\n"
```

```
      "       bulk_bench  endpoint len chunk    benchmark throughput of bulk
endpoint\n"
      "       upload      file start len        upload binary file to RAM\n"
      "       set         address byte          changes values of a single
byte\n"
      "       program     file.ihx              programs fx2 using Intel hex
format file\n"
      "\n"
      );
}

int main(int argc, char *argv[])
{
char *bus_name="001", *device_name="003";
char a;
if(argc<4){
      show_help();
      return -1;
      }
usb_init();
usb_find_busses();
usb_find_devices();
if(!strcasecmp(argv[3], "dump_busses")){
      dump_busses();
      return 0;
      }
bus_name=argv[1];
device_name=argv[2];
current_device=find_device(bus_name, device_name);
if(current_device==NULL){
      fprintf(stderr,"Cannot find device %s on bus %s\n", device_name,
bus_name);
      return -1;
      }
fprintf(stderr,"Using device %s on bus %s vendor id 0x%04x product id 0x%04x\n",
      device_name, bus_name, current_device->descriptor.idVendor,
current_device->descriptor.idProduct);
current_handle=usb_open(current_device);
if(!strcasecmp(argv[3], "dump")){
      if(argc<6){
            fprintf(stderr,"Incorrect dump command syntax\n");
            return -1;
            }
      dump_ram(atoz(argv[4]), atoz(argv[5]));
      return 0;
      }
else if(!strcasecmp(argv[3], "bulk_dump")){
      if(argc<7){
            fprintf(stderr,"Incorrect bulk_dump command syntax\n");
            return -1;
            }
      dump_bulkdata(atoz(argv[4]), atoz(argv[5]), atoz(argv[6]));
      return 0;
      }
else if(!strcasecmp(argv[3], "bulk_bench")){
      if(argc<7){
```

```
            fprintf(stderr,"Incorrect bulk_bench command syntax\n");
            return -1;
            }
      bench_bulk(atoz(argv[4]), atoz(argv[5]), atoz(argv[6]));
      return 0;
      }
else if(!strcasecmp(argv[3], "upload")){
      if(argc<7){
            fprintf(stderr,"Incorrect upload command syntax\n");
            return -1;
            }
      upload_file(argv[4], atoz(argv[5]), atoz(argv[6]));
      return 0;
      }
else if(!strcasecmp(argv[3], "set")){
      if(argc<6){
            fprintf(stderr,"Incorrect set command syntax\n");
            return -1;
            }
      a=atoz(argv[5]);
      upload_ram(&a, atoz(argv[4]), 1);
      return 0;
      }
else if(!strcasecmp(argv[3], "program")){
      if(argc<5){
            fprintf(stderr,"Incorrect program command syntax\n");
            return -1;
            }
      program_fx2(argv[4]);
      return 0;
      }
usb_close(current_handle);
return 0;
}
```

5.2 Datendurchsatz

Mit diesem Testprogramm wird die Zeit benötigt die der Controller benötigt um eine Datenmenge in den Speicher (Hauptspeicher) zu laden und dann wieder auszulesen.

```
#!/bin/bash

make || make
```

```
DESCR=`fx2_programmer any any dump_busses | grep UNCONFIGURED | head –n 1`
echo “Using device $DESCR”
BUS=`echo “$DESCR” | cut –f 3 –d \  `
DEVICE=`echo “$DESCR” | cut –f 5 –d \  `

#
# put 8051 into reset
#
fx2_programmer $BUS $DEVICE set 0xE600 1
#
# program 8051
#
fx2_programmer $BUS $DEVICE program test.ihx
#
# take 8051 out of reset
#
fx2_programmer $BUS $DEVICE set 0xE600 0
#
# dump results of the computation (which is finished by now…)
#
fx2_programmer $BUS $DEVICE bulk_dump 0x88 640 64
fx2_programmer $BUS $DEVICE bulk_bench 0x88 1048576 256
```

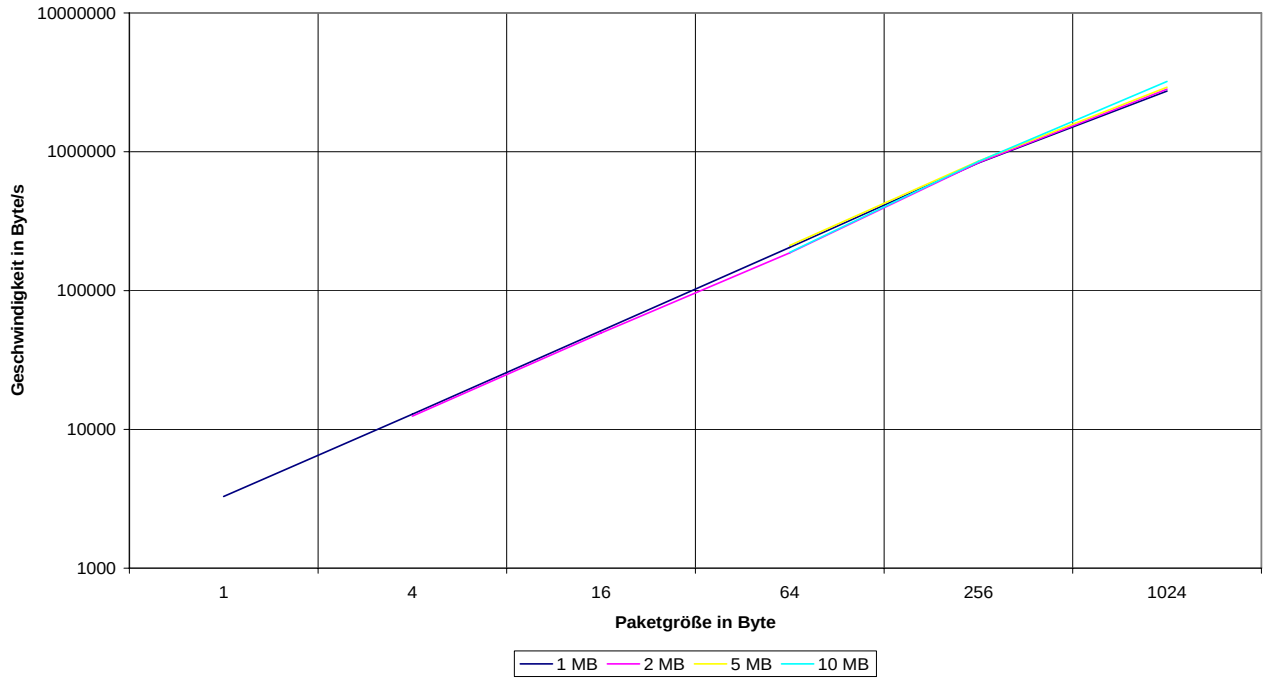
Suche nach dem USB-BUS bzw. dem USB-Device

Hier ist stets der Programmname abzuändern. In diesem fall ***test.ihx***.

Messergebnisse des Benchmarks

Blockgröße	1 MB	2 MB	5 MB	10 MB
Paketgröße	B/s	B/s	B/s	B/s
1	3290			
4	12840	12454		
16	51394	49383		
64	204504	187251	211094	188274
256	828716	835158	856053	849587
1024	2734537	2822314	2916866	3211342

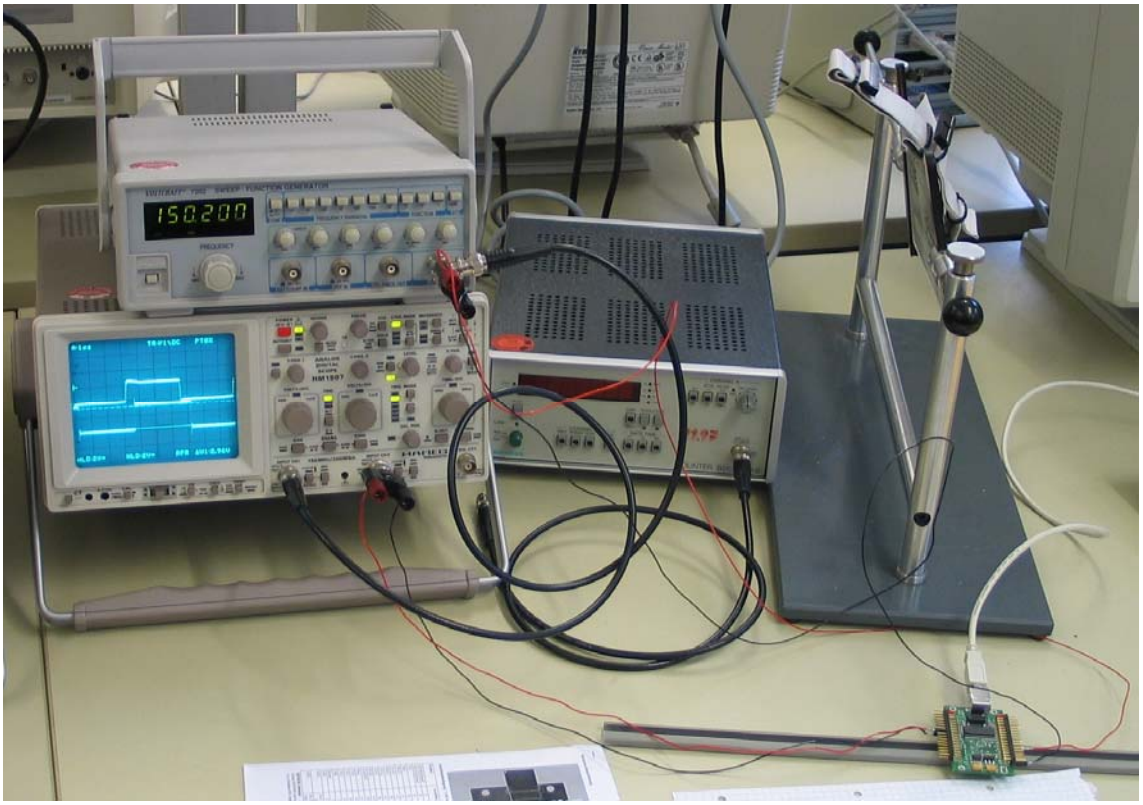
Übertragungsraten



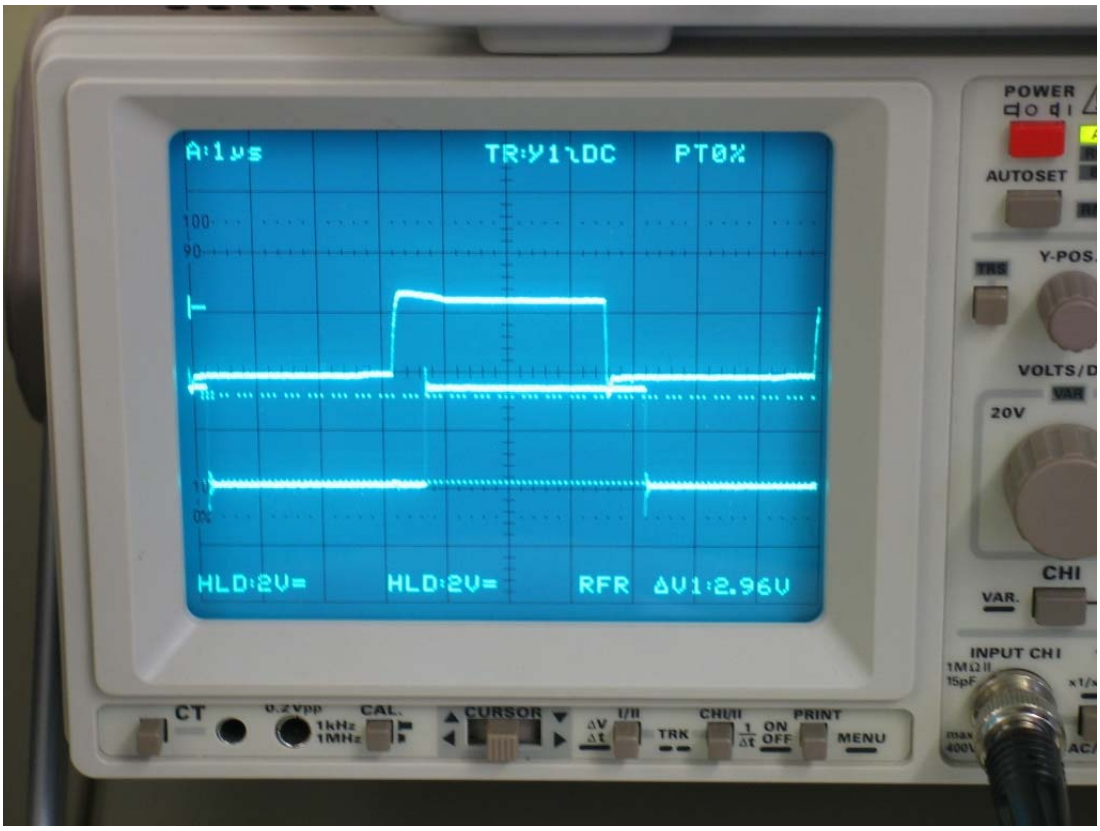

```
mc - ~/proteus/007_test - Shell No. 2 - Console
Session Edit View Bookmarks Settings Help
Programming   1 byte  starting at 0x00ee, checksum ok
Programming  12 bytes starting at 0x00ef, checksum ok
Programming  10 bytes starting at 0x00fb, checksum ok
Programming   1 byte  starting at 0x0105, checksum ok
Programming   3 bytes starting at 0x0106, checksum ok
Programming  11 bytes starting at 0x0109, checksum ok
Programming   3 bytes starting at 0x0114, checksum ok
Programming   8 bytes starting at 0x0117, checksum ok
Programming   5 bytes starting at 0x011f, checksum ok
Programming   1 byte  starting at 0x0124, checksum ok
Programming   3 bytes starting at 0x0125, checksum ok
Programming   1 byte  starting at 0x0128, checksum ok
End of file
Using device 006 on bus 002 vendor id 0x04b4 product id 0x8613
Using device 006 on bus 002 vendor id 0x04b4 product id 0x8613
0x0000: 00 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x0040: 01 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x0080: 02 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x00c0: 03 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x0100: 04 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x0140: 05 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x0180: 06 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x01c0: 07 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x0200: 08 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
0x0240: 09 00 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a
19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01
Using device 006 on bus 002 vendor id 0x04b4 product id 0x8613
Read 1048576 bytes in 0 seconds and 65983 microseconds. Rate of 15891608 bytes/second
New  Shell  Shell No. 2
```

5.3 Latenzzeit

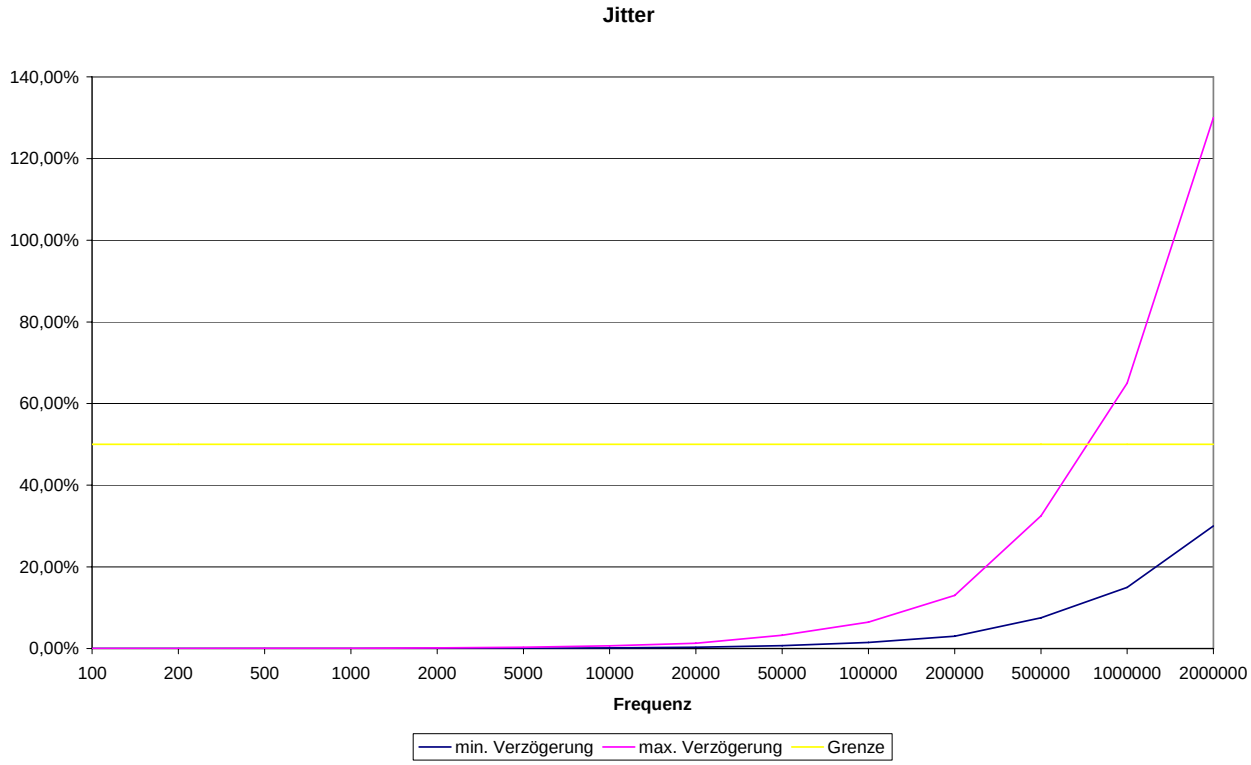
Da eine genaue Bestimmung einer Latenzzeit sehr komplexe Vorgehensweisen erfordert haben wir an einem vereinfachten Aufbau die grenzen der Signaltreue am µC direkt getestet. Mittels Frequenzgenerator wird ein Rechtecksignal auf einen IN-Port gelegt und am OUT-Port über ein Oszilloskope dargestellt (Bild 1).

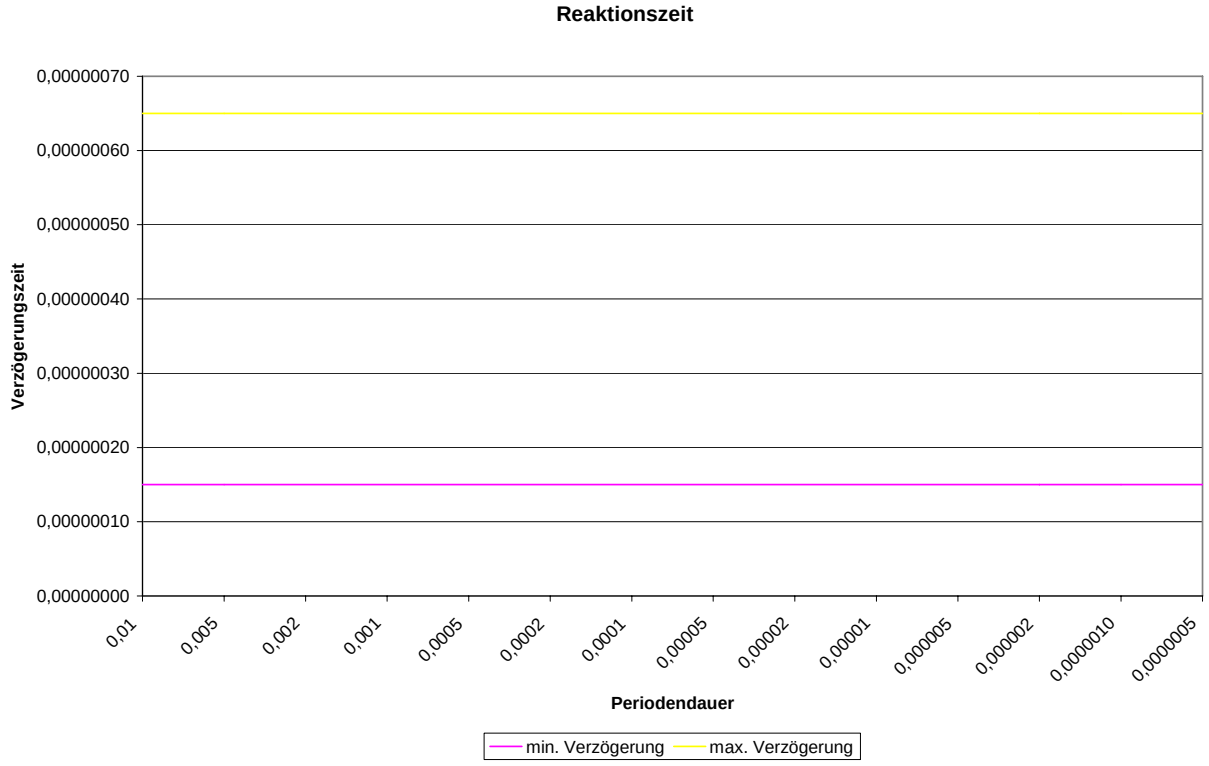


Durch langsame Steigerung der Schaltfrequenz kann man bei etwa 800 kHz erkennen dass der Jitter Effekt sich Prozentual immer stärker auswirkt (siehe auch Messtabelle). Dadurch ist ab der oben genannten Frequenz mit einem nicht optimierten Assembler Code (wie der von uns verwendete i/o Put) nicht möglich.
Bei einer stark erhöhten Frequenz z.B. 2 MHz kann man sehr gut erkennen, dass mehrere Signalblöcke übersprungen werden und somit die Ausgangsfrequenz nicht mehr mit der Eingangsfrequenz übereinstimmt. Die zeitliche Abarbeitungsverzögerung des Cotrollers, des Rechtecksignals am Ausgang sowie eine Phasenverschiebung sind auf dem Display gut sichtbar (Bild 2 & Messtabelle).



Frequenz *[Hz]*	Periodendauer *[s]*	min. Verzögerung *[ns]*	max. Verzögerung *[ns]*	min. Verzögerung *[%]*	max. Verzögerung *[%]*
100	10m	150	650	0,00	0,01
200	5m	150	650	0,00	0,01
500	2m	150	650	0,01	0,03
1k	1m	150	650	0,02	0,07
2k	500µ	150	650	0,03	0,13
5k	200µ	150	650	0,08	0,33
10k	100µ	150	650	0,15	0,65
20k	50µ	150	650	0,30	1,30
50k	20µ	150	650	0,75	3,25
100k	10µ	150	650	1,50	6,50
200k	5µ	150	650	3,00	13,00
500k	2µ	150	650	7,50	32,50
1M	1µ	150	650	15,00	65,00
2M	500n	150	650	30,00	130,00


Jitter
140,00%
120,00%
100,00%
80,00%
60,00%
40,00%
20,00%
0,00%
100
200
500
1000
2000
5000
10000
20000
50000
100000
200000
500000
1000000
2000000
Frequenz
min. Verzögerung
max. Verzögerung
Grenze


Reaktionszeit
0,00000070
0,00000060
0,00000050
0,00000040
0,00000030
0,00000020
0,00000010
0,00000000
Verzögerungszeit
0,01
0,005
0,002
0,001
0,0005
0,0002
0,0001
0,00005
0,00002
0,00001
0,000005
0,000002
0,0000010
0,0000005
Periodendauer
min. Verzögerung
max. Verzögerung

Berechnung:

Prozessor 48MHz → Periodendauer 20,8ns

Bearbeitungszyklus 12 Takte / Befehl

Verzögerung: 150ns (kleinste)
650ns (größte)

Rechnerische Taktzeit: 7,2 Takte (schnellste Schaltzeit)
31,2 Takte (langsamste Schaltzeit)
Differenz = 31,2 – 7,2 = 24 Takte

Für richtige erfassung des Signales werden 2xZyklen benötigt → 48 Takte im Mittel

Theoretische Abfragefrequenz: $\frac{1}{2{,}083 \cdot 10^{-8} \cdot 24 \cdot 2} = 1MHz$

Allerdings über Jitter (bedingt durch Programmablauf) wesentlich geringer.

Abtastfrequenz: $\frac{1}{2{,}083 \cdot 10^{-8} \cdot 31{,}2} = 1{,}52MHz$

Dadurch dass wir pro Takt zweimal Abtasten müssen, ist auf dem Frequenzgenerator nur die halbe Frequenz eingestellt: $\frac{1}{2{,}083 \cdot 10^{-8} \cdot 31{,}2 \cdot 2} = 769kHz$

6. Anhang

6.1 Resümee

Die Projektarbeit USB2 ist ein sehr Interessantes und vielseitiges Thema. Durch das Zusammenspiel der Verschiedenen Komponenten wie Mikrokontroller, Linux, C-Compiler, libusb und Ansteuerung des Interfaces ergibt sich dadurch ein breites Aufgaben- und Anwendungsfeld. Durch Aufteilung unserer Gruppe auf je zwei Personen pro Aufgabenfeld, konnten wir die verschiedenen Anforderungen gut Bewältigen.

Leider wurden unsere Anfangsversuche von vielen Fehlschlägen und falschen Fährten behindert. Da wir erst an einer Umsetzung über Windows mit der EZ-USB-Konsole festhielten. Jedoch durch Hilfestellungen und Beispieldateien aus dem Internet und besser kommentierten und ausgewerteten Materialien von Jens Dopke (Uni Wuppertal) gelang es uns endlich, das ganze System auf Linux um zusetzten.

Als dann die funktionstüchtige Version des SDDC(2.3.0) Compilers installiert wurde, stand einer „schnellen“ Weiterentwicklung unseres Projektes nichts mehr im Weg. Die vorhandenen Trials aus dem Internet und das mitgebrachte „know how“, wurden schnell neue Programme entwickelt.

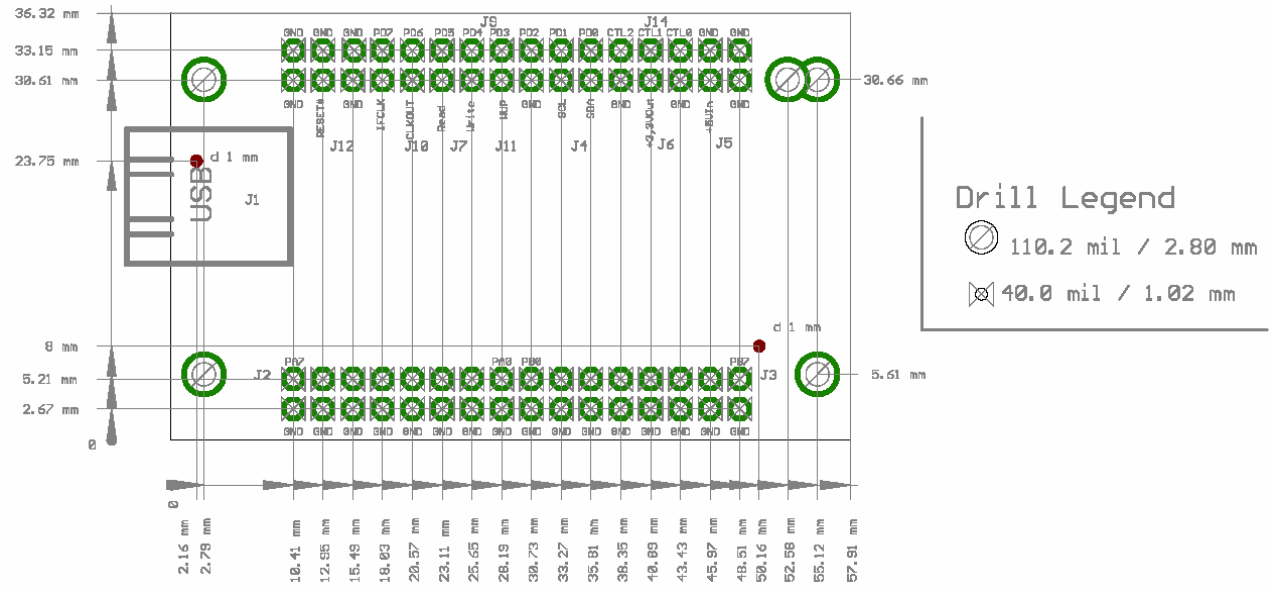
Zusammengefasst ist dies ein sehr Interessantes und Informatives Projekt, das noch viel mehr Möglichkeiten in sich birgt.

6.2 Zeitplan

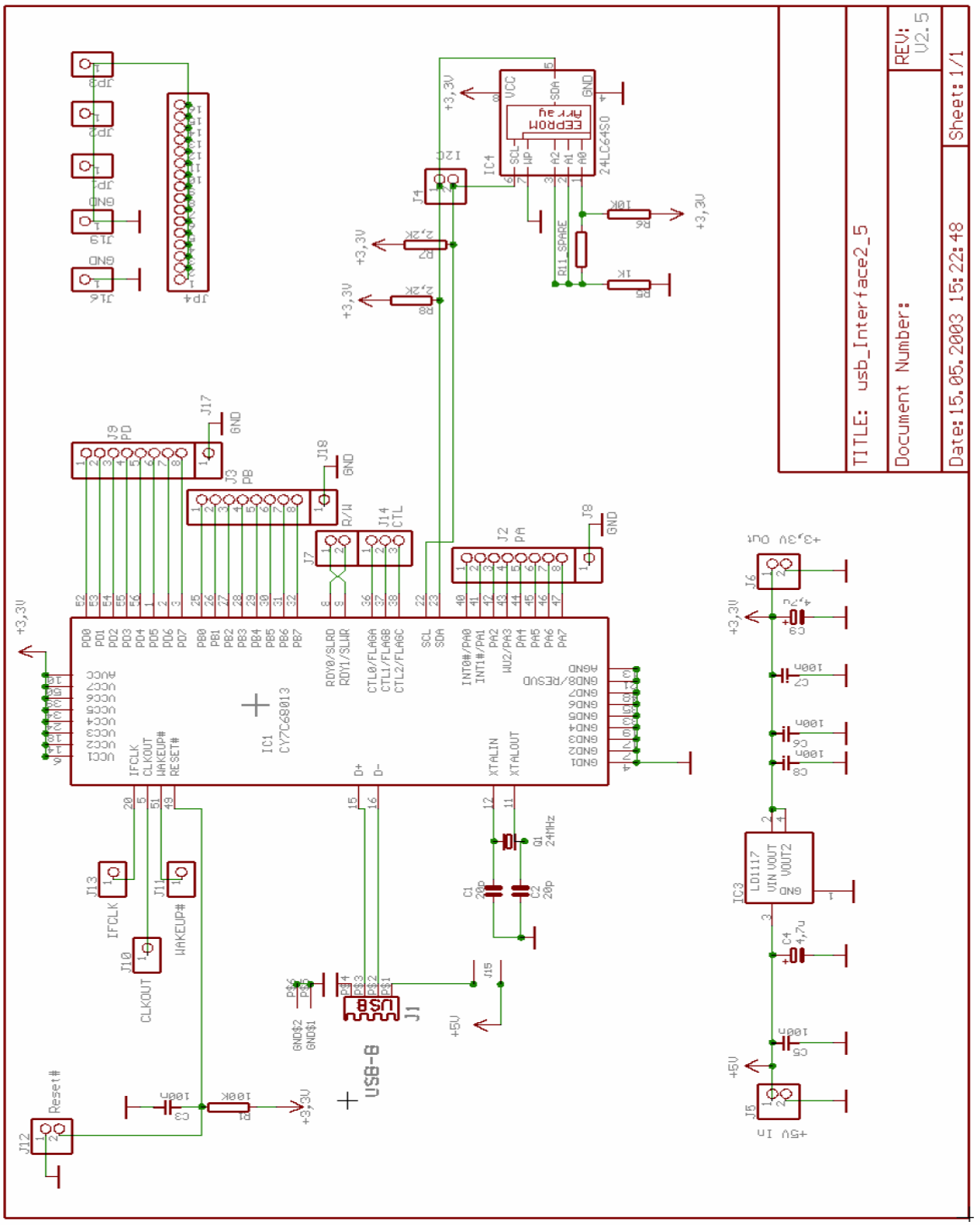
Okt 04	Nov 04	Dez 04
Informationsrecherche		
Gruppeneinteilung		
	Hardwarebeschaffung	
	Einarbeitung	
		Projektdokumentation
		Inbetriebnahme Interface

Jan 05	Feb 05	Mrz 05
Projektdokumentation		
Software Testprogramme		
	Testläufe	
	Diagnose	
		Projektabschluss

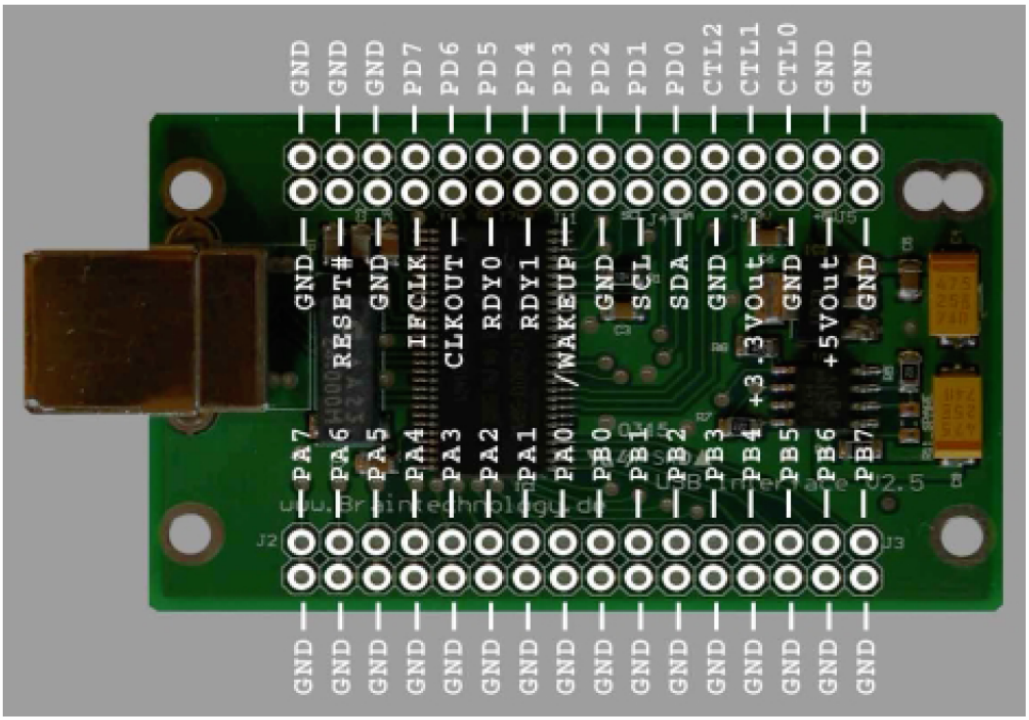
6.3 Datenblätter / Schaltpläne



Schaltplan:


TITLE: usb_Interface2_5
Document Number:
REV: V2.5
Date: 15.05.2003 15:22:48
Sheet: 1/1
IC1 CY7C68013
IC3 LD1117
IC4 24LC64S0
EEPROM Array
USB
USB-B
J1
J2 PA
J3 PB
J9 PD
J11 WAKEUP#
J12 Reset#
J13 IFCLK
J14 CTL
J10 CLKOUT
Q1 24MHz
+3,3V
+5V
+5V IN
GND

Hardware Beschreibung:



Anschlussbelegung: *(mit verwendung der USB dll)

Symbol	Anschluss	Pin Nummer	Type	Funktion
RDY0/SLRead*	J7	2	Out	Out / Lesen*
RDY1/SLWrite*	J7	1	Out	Out / Schreiben*
SCL	J4	2	Out	I²C Clock
SDA	J4	1	In/Out	I²C Daten
PB 0-7	J3	1-8	In/Out	PB 0-7
PA 0-7	J2	1-8	In/Out	PA 0-7
PD 0-7	J9	1-8	In/Out	PD 0-7
+5V In	J5	1	In	(+5V von USB offen) +5VDC Spannungsversorgung von J5 extern
+3,3V	J6	1	Out	Ausgangsspannung +3,3VDC
IFCLK	J13	1	Out	GPIF Extern Clock
CLKOUT	J10	1	Out	Clock 48 MHz
Wup	J11	1	In	Wake Up
CTL0	J14	1	Out	GPIF Control Output
CTL1	J14	2	Out	GPIF Control Output
CTL2	J14	3	Out	GPIF Control Output
USB	J1	-	In/Out	USB Connector von PC
Reset#	J12	2	In	Chip Reset
GND	J5,J6	2	-	GND
GND	J8,12,16,17, 18,19	1	-	GND
GND	JP1,JP2,JP3, JP4,J16,J19	1,1-16	-	GND

EEPROM Adresse:
Default EEPROM Adresse = 1 Dezimal, möchten Sie die Adresse auf z.b 0 Dezimal ändern, fügen Sie einen Widerstand 1K an Position R11_Spare ein, so kann dann nicht mehr vom EEPROM gebootet werden.

6.4 Projektprotokoll

6.4.1 Protokoll vom 14.10.2004

Datum : **14.10.2004**
Uhrzeit: 15:10 – 15:45; 16:30 – 17:15

Teilnehmer: Bauer Fabian, Hardt Arthur, Lopez Alexander,
Schubert Christian, Shamshoum Wael, Ziegler Steffen

Thema: Erste Vorbesprechung zur Projektarbeit - USB2 -

Aufgaben:

1. Besprechung der Aufgabenstellung mit Herr Prof.Dr.-Ing. H.R. Fehrenbach
2. Besprechung zur Organisation der Projektarbeit in der Arbeitsgruppe
3. Erstellen eines Zeitplanes sowie einer Arbeitsaufteilung

Ergebnis:

1. Nützliche Hinweise zur möglichen Vorgehensweise sowie zusätzliche Informationen zur Realisierung der Projektarbeit durch Herrn Prof.Dr.-Ing. H.R. Fehrenbach.
2. Wahl des Projektleiters sowie Verteilung erster organisatorischer Tätigkeiten.

   Projektleitung (PL) : Schubert Christian, Steffen Ziegler
   DV-Management : Fabian Bauer, Steffen Ziegler, Hardt Arthur
   Doku-Management : Lopez Alexander, Shamshoum Wael

   Aktuelle Daten werden durch das DV-Management auf http://www.home.fh-karlsruhe.de/~bafa0012/usb2 archiviert. Zusätzlich wird ein entsprechendes Projekthandbuch in Hardware angelegt.

   e-Mail Verteilung bzgl. Projektprotokolls etc. wird von Herrn Hardt Arthur organisiert.

3. Terminfestlegung für eine offene Status-Runde, jeden Donnerstag von 15:45 – ca.17:00, mit allen Beteiligten einschließlich Herr Prof.Dr.-Ing. Fehrenbach.

   Freitags findet eine separate Nachbesprechung in der Arbeitsgruppe im Zeitraum 8.00 – 11.00 Uhr statt.

   Die Besprechungen finden im Raum E010 statt.

Nächster Termin : KA,15.10.2004, 8:30 Uhr

6.4.2 Protokoll vom 15.10.2004

Datum : **15.10.2004**
Uhrzeit: 9:45 – 12:15

Teilnehmer: Bauer Fabian, Hardt Arthur, ~~Lopez Alexander~~,
Schubert Christian, Shamshoum Wael, Ziegler Steffen

Thema: Projektplanung und Aufgabenverteilung

Aufgaben:

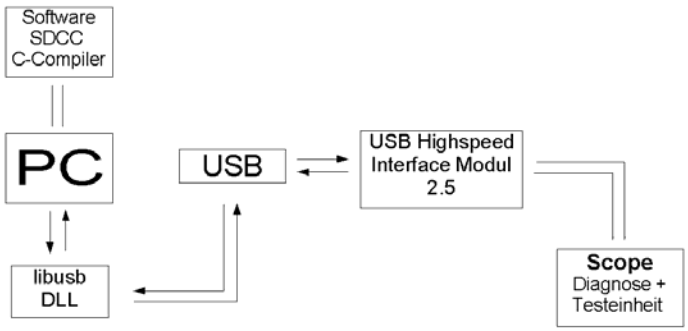
1. Zusammenfassung erster Informationen zur Projektarbeit und Definition der Zielaufgabe.
2. Verteilung der Arbeitsbereiche
3. Erstellen eines Projektplans mit Meilensteinen
4. Grobe Recherche der Hard – und Software in den einzelnen Arbeitsbereichen

Ergebnis:

1. Informationsquellen zu den einzelnen Arbeitsbereichen werden auf http://proteus.elite.to/ abgelegt.

   Verwendung von lizenzfreier Software (Open-Source)

   Die Zielaufgabe wird hier als Graphische Darstellung der USB2 – Schnittstelle präsentiert.



2. Arbeitsbereich: Programmierung - Ziegler Steffen, Schubert Christian
   Interface Modul - Bauer Fabian, Shamshoum Wael
   Diagnose Tool - Hardt Arthur, Lopez Alexander

3. Der erstellte Projektplan (Pojektplan_PROTEuS.xls) wird unter http://proteus.elite.to/ abgelegt.
   Vergabe der ersten zwei Meilensteine sowie Deadline – Termin. Weitere Planungen werden in der nächsten Besprechung ergänzt.

4. Jeder Arbeitsbereich organisiert die entsprechenden Informationen eigenständig. (Internet, etc.) Die Bestellung eventueller Arbeitsmittel (Hardware, Software) wird über Herr Ziegler Steffen organisiert.

Nächster Termin : 21.10.2004, 15:45 – 17:00 Uhr

6.4.3 Protokoll vom 21.10.2004

Datum : **21.10.2004**
Uhrzeit: 14:45-15:45-17:00

Datei: Projektdokumentation(in).doc

Teilnehmer: Bauer Fabian, Hardt Arthur, Lopez Alexander,
Schubert Christian, Shamshoum Wael, Ziegler Steffen,
Hermann Fehrenbach

Thema: **Spezifikationen für die Projektarbeit**

Aufgaben:

1. Zusammenfassung der ersten Informationsrechargen
2. Zusätzliche Informationen von Herrn Fehrenbach
3. Erweiterung des Pflichtenheftes
4. Bearbeitung des Projektplans

Ergebnis:

1. Vorschlag zum C-Compiler von Cygwin
   (für strukturierte Programmierung)

   Hardware: Externe Platine mit Leuchtdiodenzeile 16-Bit
   (Europa-Karte, Werkstatt Hr. Helmstetter, Hr. Wäldle)

   Interface-Karte wurde bereits von Hr. Fehrenbach bestellt. Zusätzliche Informationen findet man im Internet, etc. (Suche nach bereits ähnlichen Projekten)

   Datenblatt, Funktionsbeschreibung zum 8051-MC organisieren

   Scope - Funktion mittels Oszilloscope , Zeitmessung des Datendurchsatz, Latenzzeitmessung Suche im Internet.

2. Über zusätzliche Informationen wurde bereits im allgemeinen Informationsaustausch gesprochen. Eventueller Ansprechpartner Hr. Geggus.

3. Das Pflichtenheft wird um den Punkt Randbedingungen (PC-Testanlage) ergänzt.

4. Verlegung des Meilensteins M1 auf KW44 sowie Erfassung der wichtigsten Informationen.

Nächster Termin : 28.10.04, 15:45 Uhr

6.4.4 Protokoll vom 28.10.2004

Datum : **28.10.2004**
Uhrzeit: 14:00 – 15:30

Teilnehmer: Bauer Fabian, Hardt Arthur, Lopez Alexander, ~~Schubert Christian~~, Shamshoum Wael, Ziegler Steffen, ~~Hermann Fehrenbach~~

Thema: **Spezifikationen für die Projektarbeit**

Aufgaben:

5. Zusammenfassung neusten Informationsrecherchen

Ergebnis:

5. Weitere Differenzierung der Impulszeitenauswertung bzw. Latenzzeitmessung. Lösungsansätze diskutiert wie man die Latenzzeitmessung mittels Oziloskop und Frequenzgenerator, ebenso den Datendurchsatz, ermittelt werden kann.

   Datenblatt und Funktionsbeschreibung zum 8051-MC wurden auf D:\Proteus\ hinterlegt.

   Linux-Rechner:
   - Xandros - Linux installiert.
   - Hardwareoptimierung durch einbauen von zusätzlichem Arbeitsspeicher.
   - USB2 Karten eingebaut.

   Windows XP Rechner:
   - Windows XP installiert.
   - USB2 Karten besorgt und eingebaut, Treiber Installiert. Nach dem Einbau läuft Windowsrechner instabil, beim benutzen der USB-Schnittstelle bleibt das System hängen verm. durch IRQ Interrupt Fehler. Fehlersuche und Beseitigung.
   - LibUSB 0.1.8.10 installiert und erfolgreich über ein Testprogramm getestet.
   - SDCC installiert
   - mit Cygwin beschäftigt, noch nicht komplett zur Funktion gebracht

   Allgemeine Informationen:

   Werkzeug kann bei Herrn Wäldle vormittags abgeholt werden (Werkzeugkasten, abschließbar)

   Für den Raum LI U09 haben Arthur Hardt, Wael Shamshoum und Christian Schubert einen Schlüssel.

Nächster Termin : 04.11.04, 15:45 Uhr

6.4.5 Protokoll vom 04.11.2004

Datum : **04.11.2004**
Uhrzeit: 12:00 – 14:00; 15:45 – 17:00

Teilnehmer: Bauer Fabian, Hardt Arthur, Lopez Alexander,
Schubert Christian, Shamshoum Wael, Ziegler Steffen,
Hermann Fehrenbach

Thema: **Arbeitsplanung in den Aufgabenbereichen und weiterführender Tätigkeiten**

Aufgaben:

6. Zusammenfassung der neusten Informationsrecherchen
7. Besprechung der weiteren Tätigkeiten in den Aufgabenbereichen
8. Zusätzliche Organisationen

Ergebnis:

1. Software: Installation von DevC++ -Compiler auf Windows.
   Unterstützung für Installation unter Linux von Hr. Fehrenbach
   InterfaceCard und Diagnose: Latenzzeit über Oszi - Messung am Ein- und Ausgang,
   Einkanalmessung für das Senden an das Interface einer bestimmten Frequenz in einem festen Zeitraster. Echtzeit (10-20ms besser kleiner)

2. Aufgaben für nächste Woche:

- Einarbeitung in DevC++, Erstellung eines make-files für Linux
- Herstellung der Versuchsplatine
- Weitere Recharge für Latenzzeiterfassung

3. Absprache mit Hr. Gäntner zur Labornutzung

Nächster Termin : 11.11.04, 15:45 Uhr

6.4.6 Protokoll vom 11.11.2004

Datum : **11.11.2004**
Uhrzeit: 15:15 – 17:00

Teilnehmer: Bauer Fabian, Hardt Arthur, ~~Lopez Alexander~~,
Schubert Christian, ~~Shamshoum Wael~~, Ziegler Steffen, Hermann Fehrenbach

Thema: **Spezifikationen für die Projektarbeit**

Aufgaben:

9. Zusammenfassung neusten Informationsrecherchen

Ergebnis:

1. Platinen und Bauteile für die Auswertungseinheit bestellt
2. Software:
   8051-Maschine (Emulator) installiert
   SDCC C-File umgewandelt in 8051-Machine File umgewandelt und simuliert
3. Aufhaben für die Nächste Woche:
   - Programmierung der LIBUSB und Ansteuerung eines Ports am USB-Interface
   - Bestückung der Testplatine und Versuchaufbau

Nächster Termin : 18.11.04, 15:45 Uhr

6.4.7 Protokoll vom 25.10.2004

Datum : **25.11.2004**
Uhrzeit: 15:30 – 17:00

Teilnehmer: Bauer Fabian, Hardt Arthur, Lopez Alexander,
Schubert Christian, Shamshoum Wael, Ziegler Steffen, Hermann Fehrenbach

Thema: **Spezifikationen für die Projektarbeit**

Aufgaben:

10. Zusammenfassung des aktuellen Projektstands
11. Absprache zum weiten Vorgehen zur Inbetriebnahme des USB-Interfacemoduls
12. Projektunterstützung nächste Woche
13. Weiterführende Tätigkeiten der Projektgruppen

Ergebnis:

4. Hardware ist fertig und muss dokumentiert werden
5. Erprobung der Programmiersoftware AVR Studio und der USB2.0.dll
6. Herr Geggus wird nächste Woche Support zum Projekt geben
7. Aufhaben für die Nächste Woche:
   - Informationsrecherche zum USB2.0 und I²C Bus - Protokoll
   - Erstellen einer technischen Dokumentation für Hard -und Software
   - Erstellen einer Powerpoint Präsentation
   - Telefonische Information zu SDCC und Inbetriebnahme des Interfacemodules bei der Fa. Braintechnology und Uni Wuppertal
   - Erweiterung des Pflichtenheftes

Nächster Termin : 1.12.04, 15:45 Uhr

6.4.8 Protokoll vom 02.12.2004

Datum : **02.12.2004**
Uhrzeit: 15:30 – 18:00

Teilnehmer: Bauer Fabian, Hardt Arthur, Lopez Alexander,
Schubert Christian, Shamshoum Wael, Ziegler Steffen, Hermann Fehrenbach

Thema: **Spezifikationen für die Projektarbeit**

- Besuch von Herr Geggus zur Projektunterstützung

Aufgaben:

14. Inbetriebnahme des USB-Interfacemoduls
15. Versuch mit FX-Load unter Linux (mit Kernel 2.3) die Firmware herunter zu laden
16. Tests von Programmen (Herr Geggus)

Ergebnis:

8. Download der Firmware misslungen
9. Diverse Tests blieben Erfolglos
10. Überlegungen zur weiteren Vorgehensweiße

Nächster Termin : 09.12.04, 15:45 Uhr

6.4.9 Protokoll vom 09.12.2004

Datum : **09.12.2004**
Uhrzeit: 15:30 – 18:00

Teilnehmer: Bauer Fabian, Hardt Arthur, Lopez Alexander,
Schubert Christian, Shamshoum Wael, Ziegler Steffen, Hermann Fehrenbach

Thema: **Spezifikationen für die Projektarbeit**

Aufgaben:

17. Zusammenfassung des aktuellen Projektstands
18. Absprache zum weiten Vorgehen zur Inbetriebnahme des USB-Interfacemoduls
19. Projektunterstützung nächste Woche
20. Weiterführende Tätigkeiten der Projektgruppen

Ergebnis:

11. Übertragung mit Cypress Developementtool ist nicht ohne weiteres möglich. Verschiedene Anfragen und Hilfe angefordert (Braintechnology).
12. Cypress Developementtool ausprobiert und getestet.
13. Nach Möglichkeiten gesucht C-Files über die Entwicklungstools auf das Interface zu übertragen.
14. Verschiedene Tutorials zum Cypress Developementtool ausprobiert.
15. Xandros auf Rechner Installiert
16. Kontakt mit anderen Projektgruppen aufgenommen, die mit selbem Interface Arbeiten

Nächster Termin : 16.12.04, 15:45 Uhr

6.5 Spezifikation

Spezifikation

Zur Projektarbeit USB2.0

Aufgabenstellung: **Inbetriebnahme eines USB - Diagnose Tools**

Projekt: **PROTEuS**

Kunde: **FH-Karlsruhe**

Version: **1.0**

Ausgabedatum: **15.10.2004**

Auftragsumfang

Es soll eine USB2-Prototypenplatine beschafft und in betrieb genommen werden. Mit Hilfe eines Testaufbaus sollen erreichbare Datentransferraten und Latenzzeiten bestimmt werden.

Ablaufbeschreibung

Randbedingungen

PC-Aufbau

Soll-Ablauf:

1. Beschaffung und Inbetriebnahme der Prototypenplatine
2. Inbetriebnahme des freien C-Compilers für 8051 – Mikrokontroller
3. Inbetriebnahme und Test der Programmbibliothek „ libusb“
4. Entwurf und Realisierung eines Testaufbaus für die Ermittlung der erreichbaren I/O Leistung

Auftragsbeschreibung

Zur Realisierung von universellen, kostengünstigen und schnellen I/O Schnittstellen hat sich der USB – Bus hervorragend bewährt. USB2.0 bietet darüber hinaus hohe Datentransferraten. Inzwischen gibt es Interface-Bausteine mit angekoppelten 8051-kompatiblem Mikrokontroller in Form von Prototypenplatinen.
Für die Datenkommunikation steht eine freie (LPGL). Plattformübergreifende Programmbibliothek zur Verfügung.
Zur Programmierung des 8051-kompatieblen Mikrokontroller kann ein freier C-Compiler benutzt werden.
Es soll eine USB2-Prototypenplatine beschafft und Inbetrieb genommen werden.
Mit Hilfe eines Testaufbaus sollen erreichbare Datentransferraten und Latenzzeiten bestimmt werden.

Verwendete Info-Links:

- http://proteus.elite.to/
- http://www.braintechnology.de/braintechnology/
- http://libusb.sourceforge.net/index.html
- http://libusb-win32.sourceforge.net/
- http://sdcc.sourceforge.net/
- http://www.atlas.uni-wuppertal.de/~dopke/wodan2/ezusb.html

Projekttermin

Auftragsvergabe:	KW 42
Beginn Vorbereitung:	KW 42, KW43
Erstellung Pflichtenheft:	KW 44
Inbetriebnahme:	KW 51
Endabnahme:	KW 52

Ansprechpartner

Bereich	Name	Abt.	Telefon
Projektorganisation & Programmierung	Hr. Schubert Hr. Ziegler	PL / SW - Dev.	0172 710 2193 0173 487 4482
DV - Management & Interface - Modul	Hr. Bauer Hr. Shamshoum	DV / IF - Dev.	0172 778 3981 0179 866 9170
Doku – Management & Diagnose Tool	Hr. Hardt Hr. Lopez	DOKU / Diag – Dev.	0172 714 8255 0172 916 4156

7. CD

Auf dieser CD sind enthalten:

- SDCC 2.3.0
- Fx2-programmer
- Beschreibung des Interface
- Beschreibung des 8051
- Trials-Programme
- Waloda-ext-Programme
- Proteus-Programme