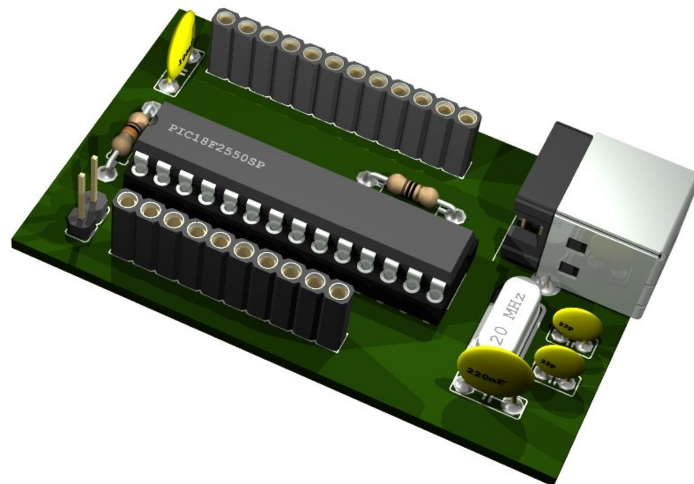
USB4all

V.9

Technische Beschreibung
und Nutzungsanleitung



Autor: sprut
Stand: 22.01.2012

1 Inhaltsverzeichnis

1 Inhaltsverzeichnis.....2
2 Abbildungsverzeichnis.....4
3 Nutzungsbedingungen.....5
4 Einleitung.....5
5 Hardware.....6
5.1 USB-Beschaltung des Chips.....6
5.2 Anderer Takt mittels Bootloader.....8
5.3 Anderen Takt beim Brennen einstellen.....8
5.4 Nutzbare Interfaces.....10
5.4.1 Ausgangspins.....11
5.4.2 Eingangspins.....13
5.5 TTL-IO-Port-Anschluss.....13
5.6 ADC-Anschluss.....13
5.7 Zählfrequenzmesser-Anschluss.....14
5.8 RS232-Anschluss.....14
5.9 I2C-Anschluss.....14
5.10 LCD-Anschluss.....15
5.11 PWM-Anschluss.....16
5.12 Schrittmotor-Anschluss.....16
5.12.1 Schrittmotoransteuerung mit ABCD-Phasen.....16
5.12.2 Schrittmotoransteuerung mit L297.....17
5.13 Servo-Anschluss.....17
6 Software.....18
6.1 USB-Device.....18
6.2 PC.....18
6.2.1 Treiberinstallation.....18
6.2.2 Installation: Microchip Custom Driver 19
6.2.3 Installation: USB RS-232 Emulation Driver.....25
7 Befehlsübersicht.....29
7.1 TTL-IO-Pins.....31
7.2 10-Bit-ADC.....34
7.3 Zählfrequenzmesser.....36
7.4 RS-232.....40
7.5 I2C-Interface.....42
7.6 SPI-Interface.....44
7.7 Microwire (noch nicht getestet).....47
7.8 Schieberegister-Interface.....48
7.9 LCD-Interface.....50
7.10 PWM1 und PWM2.....53
7.11 interner EEPROM.....55
7.12 Schrittmotoransteuerung.....56
7.12.1 Schrittmotorenansteuerung mit ABCD-Phasen.....56
7.12.2 L297-Schrittmotorenansteuerung.....61
7.13 Servos.....64
7.14 Impulszähler.....66
7.15 Reset des USB4all.....68
8 Ansteuerung des USB4all 69

8.1 USB4all-CDC..69
8.2 USB4all-MCD..71
8.3 Beispiele für die Ansteuerung des USB4all..72
8.3.1 Beispiel: Schreiben eines Bytes in den EEPROM..72
8.3.2 Beispiel: Messen einer Spannung...72
8.3.3 Beispiel: Messen einer Frequenz...73
8.3.4 Beispiel: Ausgabe von “Hallo” am LCD-Display...73
8.3.5 Beispiel: Einschalten einer LED am Pin RC0..73
8.3.6 Beispiel: Drehen eines Schrittmotors..74
8.3.7 Beispiel: Temperaturmessung mit einem LM75 via I2C.....................................75
8.3.8 Beispiel: Reset des USB4all..75
8.4 Nutzung des USB4all-MCD unter Linux..76
9 Bootloader..77
9.1 Start des Bootloaders..77
9.1.1 Aktivieren des Bootloaders per Software..77
9.1.2 Aktivieren des Bootloaders mit dem Jumper JP1...78
9.1.3 Neue Firmware in den USB4all laden...79
9.1.4 Falsches HEX-File in den USB4all geladen...80
10 Fehlersuche bei USB-Geräten...81
10.1 Allgemein..81
10.2 Treiber und Device (Windows)...81
10.3 Anschluss am PC...81
10.4 Typische Fehlerbilder...82

2 Abbildungsverzeichnis

Abbildung 1 USB4all-Übersicht....5
Abbildung 2 Typische Beschaltung des USB4all....7
Abbildung 3 Minimale Beschaltung des USB4all....7
Abbildung 4 Takterzeugung im Steuer-PIC....8
Abbildung 5 Resonator/Quarz-Einstellung für den Steuer-PIC....9
Abbildung 6 Pinbelegung des PIC18F2455....11
Abbildung 7 Ausgangspin - Prinzip....11
Abbildung 8 High-Pegel bei verschiedenen Ausgangsströmen....12
Abbildung 9 Low-Pegel bei verschiedenen Ausgangsströmen....12
Abbildung 10 RS232-Anschluss....14
Abbildung 11 I2C-Anschluss....15
Abbildung 12 LCD-Anschluss....15
Abbildung 13 einfacher Schrittmotoranschluss....16
Abbildung 14 Modellbauservo....17
Abbildung 15 Neue Hardware gefunden....19
Abbildung 16 Hardware Assistent 1....20
Abbildung 17 Hardware Assistent 2....20
Abbildung 18 Hardware Assistent 3....21
Abbildung 19 Gerätetreiber auswählen 1....22
Abbildung 20 Pfad zum Treiber einstellen....22
Abbildung 21 Gerätetreiber auswählen 2....23
Abbildung 22 Treiber kann installiert werden....23
Abbildung 23 Assistent fertig stellen....24
Abbildung 24 Gerätemanager....24
Abbildung 25 Gerätemanager - Energiesparoption....25
Abbildung 26 Beginn der Treiberinstallation....26
Abbildung 27 manuelle Auswahl des Treiber benutzen....26
Abbildung 28 den Pfad zur inf-Datei angeben....27
Abbildung 29 den Windows-Logo-Test ignorieren....27
Abbildung 30 Ende der Installation....28
Abbildung 31 USB4all-CDC als emuliertes COM3-Port....28
Abbildung 32 Halbschrittmode....59
Abbildung 33 Vollschrittmode....59
Abbildung 34 Wavemode....59
Abbildung 35 USBoot möchte den Bootloader aktivieren....78
Abbildung 36 Neue Firmware in den USB4all laden....79
Abbildung 37 Neue Firmware geladen....79
Abbildung 38 USB-Grundbeschaltung eines PIC....82

3 Nutzungsbedingungen

DIE SOFTWARE DARF OHNE ENTRICHTUNG EINER LIZENZGEBÜHR BENUTZT WERDEN. DAS GILT FÜR DIE PRIVATE UND GEWERBLICHE NUTZUNG.

DIE PUBLIKATION DER SOFTWARE ERFOLGT "AS IS". FÜR DIE EINHALTUNG ZUGESICHERTER EIGENSCHAFTEN ODER FÜR SCHÄDEN, DIE DURCH DEN EINSATZ ENTSTANDEN SEIN KÖNNTEN, ÜBERNIMMT DER AUTOR KEINERLEI HAFTUNG. SIE NUTZEN DIE SOFTWARE AUF EIGENE GEFAHR!

4 Einleitung

Der USB4all ist eine Fertiglösung zum Anschluss von einfachen Geräten (z.B. Bastelprojekten) an den PC mit Hilfe des USB-Busses. Es ist ein PIC18F2455 (von Microchip) mit einer Firmware, die die Interfaces des PIC18F2455 auf einfachem Wege zur Verfügung stellt, ohne dass der Nutzer Kenntnisse über den USB-Bus oder PIC-Mikrocontroller haben muss.

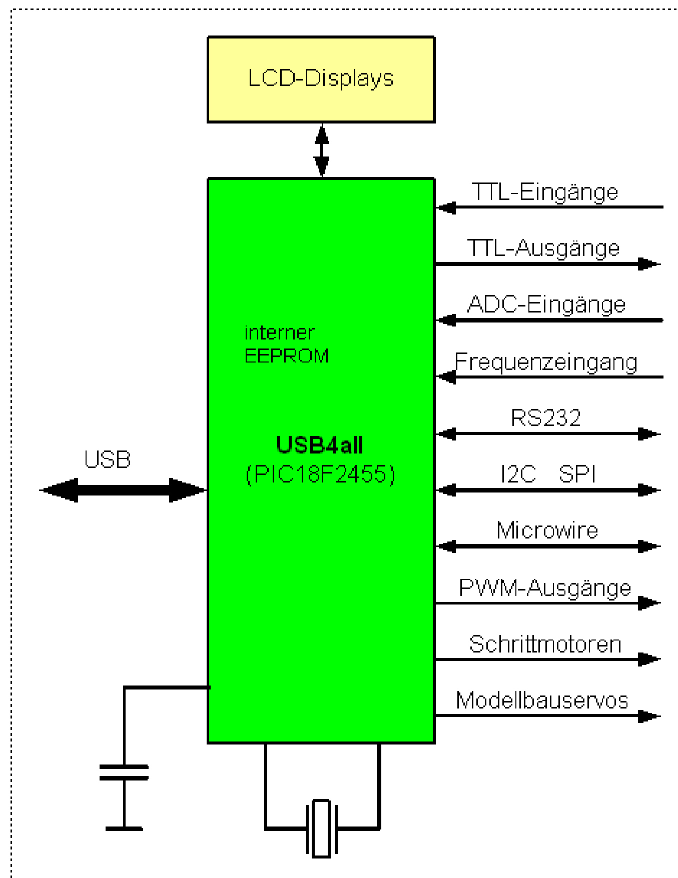


Abbildung 1 USB4all-Übersicht

Es gibt den USB4all in zwei Versionen. Der USB4all-MCD wird mit Hilfe einer speziellen DLL angesteuert. Den USB4all-CDC steuert man einfach über eine virtuelle RS232-Schnittstelle.

Die USB4all-Firmware befindet sich noch in der Entwicklungs- und Testphase. Ein großer Teil der Funktionen sind bereits einsetzbar, an anderen muss noch gearbeitet werden. Die Firmware ist für die Zusammenarbeit mit einem Bootloader ausgelegt, sie kann aber

auch autonom eingesetzt werden.

Gegenwärtig stellt USB4all dem Anwender folgende Anschlüsse zur Verfügung, die per USB angesteuert werden können:

- 20 digitale Input/Output Pins (TTL)
- 10 analoge Spannungsmesseingänge (0..5V)
- ein Frequenzmesseingang (bis 50 MHz)
- ein RS232-Anschluss
- ein I2C-Master-Anschluss
- ein SPI-Anschluss mit bis zu 6 Slave-Select-Leitungen
- ein Microwire-Anschluss
- ein Schieberegister-Anschluss
- Anschlüsse für 2 LCD-Dotmatrix-Displays
- 2 PWM-Ausgänge
- 4 Schrittmotoranschlüsse für ABCD-Phasen-Anschluss
- 4 Schrittmotoranschlüsse für L297-Schaltkreise
- Anschlüsse für 13 Modellbauservos.
- 2 Impulszähleingänge
- 192 Byte interner EEPROM-Speicher

5 Hardware

5.1 USB-Beschaltung des Chips

Der USB4all ist ein PIC18F2455 mit der von mir bereitgestellten Firmware. Alternativ kann man auch den PIC18F2550 verwenden.

++Hinweis++
In mehreren Abbildungen in diesem Dokument wird der PIC18F2550 verwendet, es kann stattdessen aber immer der PIC18F2455 eingesetzt werden.
Die Software funktioniert auch ohne Änderungen in PIC18F4455/4550 (je nach Gehäuse 40 bzw 44 Pins), allerdings werden deren zusätzliche Pins nicht verwendet und die Pin-Nummern stimmen nicht mit denen in dieser Dokumentation überein.

Um mit dem USB4all ein USB-Device aufzubauen, wird typischerweise die unten abgebildete Beschaltung verwendet.

- Mit L2 und C8 wird die Betriebsspannung für den Chip gefiltert.
- C1 dient der Stabilisierung der im Chip erzeugten 3,3V, die für das USB-Interface benötigt werden.
- R2 und JP1 dienen zur Aktivierung des Bootloaders. Wird auf JP1 ein Jumper gesteckt, und dann des Gerät an den USB-Port eines PC angeschlossen, dann startet der Bootloader, der die Aktualisierung der Firmware ermöglicht. Ohne den Jumper startet die USB4all-Firmware.
- Q1, C2 und C3 stellen den Takt für den Chip zur Verfügung.

Der Jumper JP1 dient als Backup zur Aktivierung des Bootloaders.

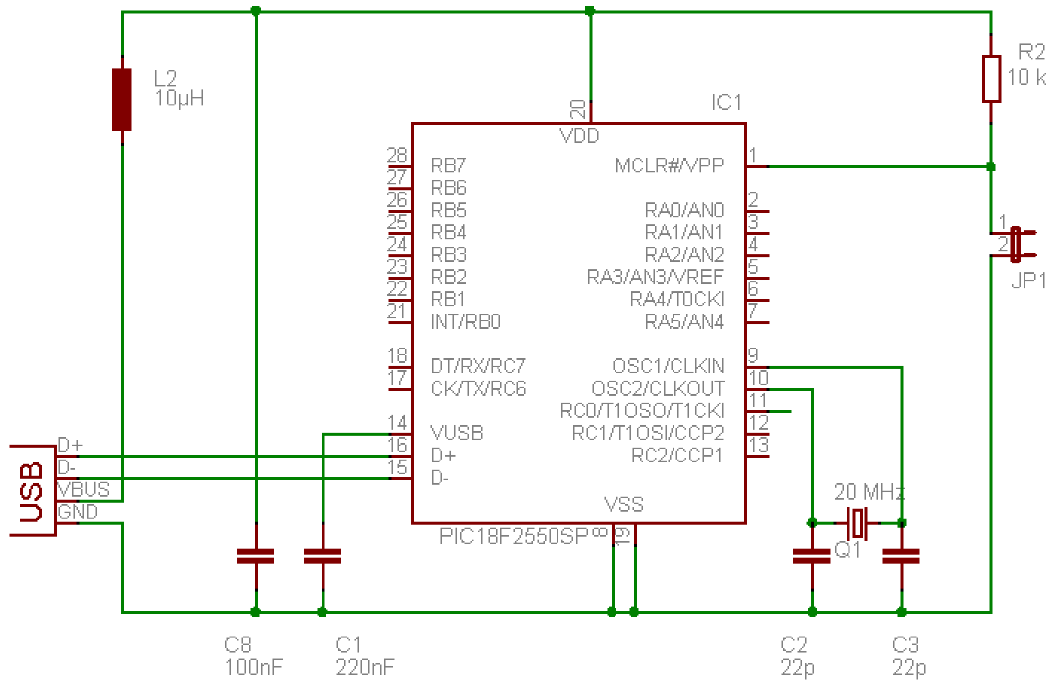


Abbildung 2 Typische Beschaltung des USB4all

Das Titelbild dieses Dokuments zeigt eine solche Beschaltung, bei der alle nutzbaren Pins auf Buchsenleisten gelegt wurden.
Die Beschaltung lässt sich noch etwas vereinfachen, wie die nächste Abbildung zeigt.

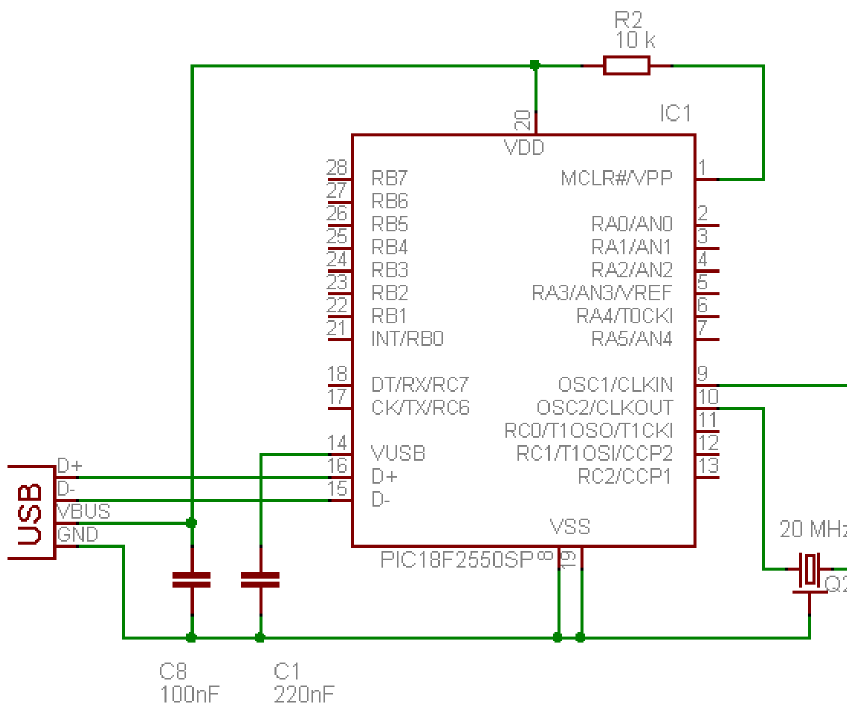


Abbildung 3 Minimale Beschaltung des USB4all

Der USB4all kann sowohl mit einem Keramikresonator wie auch mit einem Quarz betrieben werden. Wird ein Quarz verwendet, dann sind auch die beiden Lastkondensatoren für den Quarz (C2 & C3) einzusetzen. Wird dagegen ein Keramikresonator eingesetzt, dann entfallen die beiden Kondensatoren.

++ACHTUNG++
Der Einsatz eines Keramikresonators begrenzt die Genauigkeit des Frequenzmesseinganges auf 0.5%.

Als Frequenz für den Resonator/Quarz ist 20 MHz vorgesehen. Leider sind 20MHz-Resonatoren nicht leicht zu beschaffen. Der Einsatz anderer Typen ist möglich, wenn folgendes beachtet wird:

5.2 Anderer Takt mittels Bootloader

Der Königsweg zur Installation der USB4all-Firmware geht über den Bootloader-5 (ist im ZIP-File des USB4all mit enthalten). Er ermöglicht es später unproblematisch die Firmware zu aktualisieren, und er übernimmt auch die Kontrolle über den Taktgenerator des PIC.
Der Bootloader liegt in verschiedenen Varianten für verschiedene Quarzfrequenzen vor (4, 8, 20 MHz).
Man wählt einfach den passenden Bootloader aus, und brennt ihn mit einem Programmiergerät in den PIC.
Damit wird automatisch auch die Taktfrequenz für die Firmware festgelegt. Wenn man z.B. einen 8-MHz-Resonator verwenden will, dann brennt man einen 8-MHz-Bootloader in den PIC. Wird dann später die (eigentlich für 20 MHz ausgelegte) Firmware mit Hilfe des Bootloaders in den PIC geladen, dann funktioniert diese auch mit dem 8 MHz-Resonator.

++Achtung++
Als Bootloader ist der Bootloader-5 einzusetzen. Bei Verwendung eines anderen Bootloaders kann es zu Problemen mit dem 2. PWM-Kanal kommen.

5.3 Anderen Takt beim Brennen einstellen

Falls man den Bootloader nicht verwenden möchte (was für mich schwer zu verstehen wäre), lässt sich trotzdem ein anderer Quarz verwenden, wenn man die Konfiguration des PIC verändert:

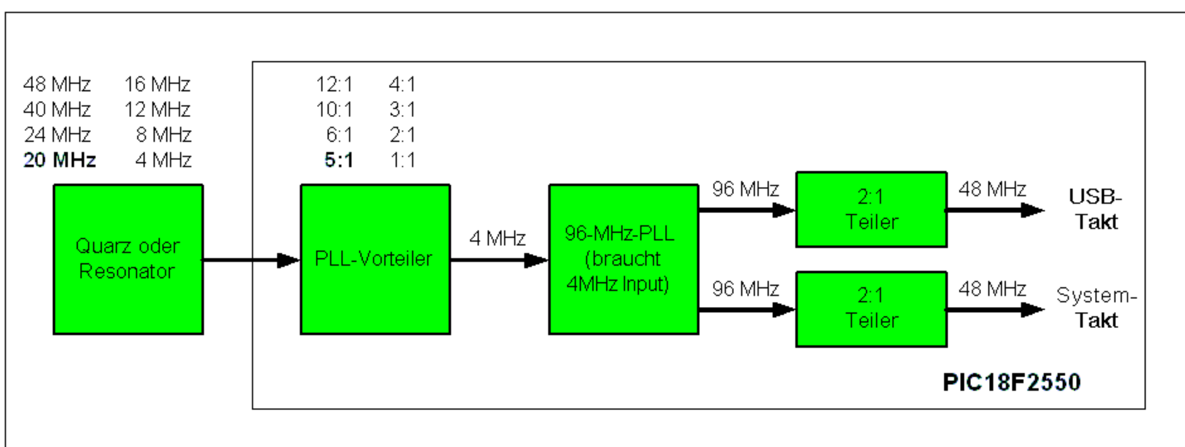


Abbildung 4 Takterzeugung im Steuer-PIC

Standardmäßig wird die Quarzfrequenz im PIC zunächst mit einem 5:1 Frequenzteiler auf

4 MHz herunter geteilt. Aus diesen 4 MHz werden anschließend mit einer PLL 96 MHz erzeugt. Diese wiederum dient als Basis für den USB-Takt (2:1 Teilung) und den PIC-Takt (ebenfalls 2:1 Teilung).

Die 4 MHz für die PLL lassen sich natürlich nicht nur aus 20 MHz erzeugen. Da der Eingangsteiler neben dem Teilverhältnis 5:1 auch die Teilverhältnisse 12:1, 10:1, 6:1, 4:1, 3:2, 2:1 und 1:1 beherrscht, kommen auch Resonatoren/Quarze mit 48 MHz, 40 MHz, 24 MHz, 16 MHz, 12 MHz, 8 MHz und 4 MHz in Frage. Man muss nur die Vorteilereinstellung ändern.

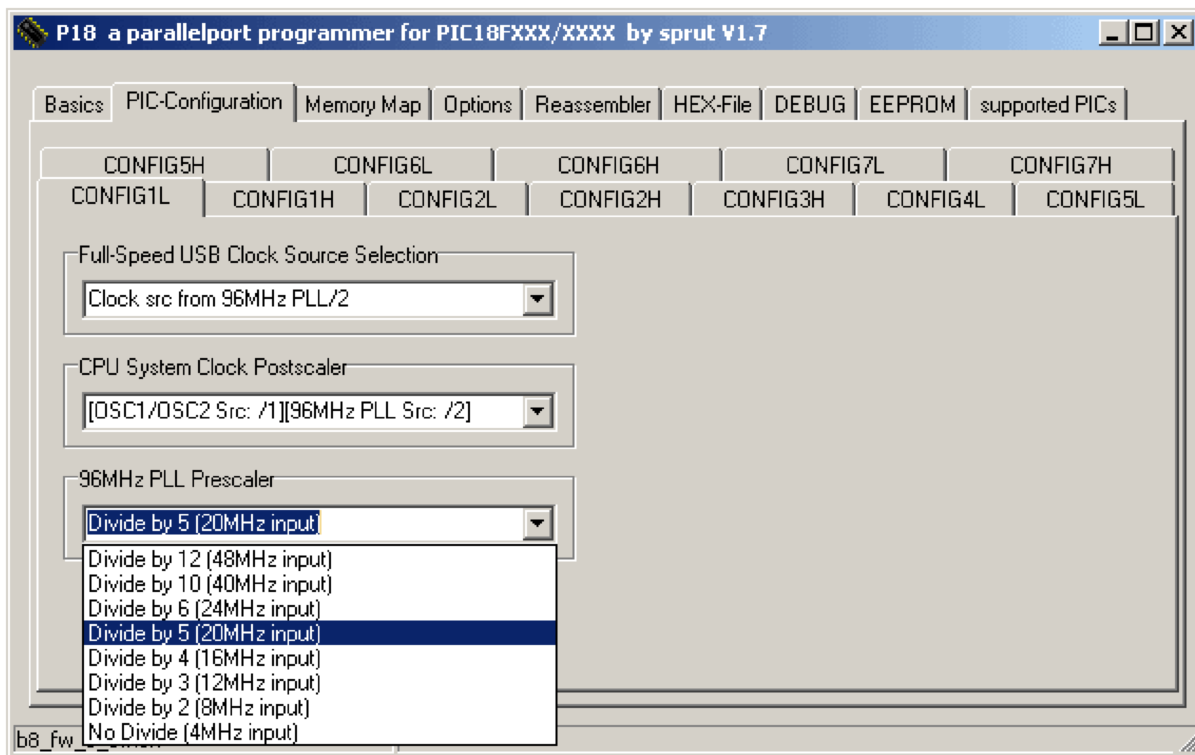


Abbildung 5 Resonator/Quarz-Einstellung für den Steuer-PIC

Die Vorteilereinstellung ist in der PIC-Konfiguration festgelegt, die im Firmware-HEX-File abgelegt ist. Mit einer geeigneten Brennsoftware, wie P18 oder US-Burn, lässt sich diese Konfigurationseinstellung vor dem Brennen des Steuer-PICs manipulieren. Dazu wird nach dem Laden des HEX-Files in der „Basics“ -Karteikarte von P18 oder US-Burn die Option „**Config from HEX-File**“ deaktiviert. Danach wechselt man auf die Karteikarte „**PIC Configuration**“. Dort wählt man die Unterkarteikarte „**CONFIG1L**“.
Hier nun findet man alle Takteinstellungen des PIC. Wichtig ist die Option „**96 MHz PLL Prescaler**“, die standardmäßig auf „**Divide by 5 (20MHz input)**“ steht. Diese Option passt man einfach der gewünschten Resonator/Quarz-Frequenz an. Die Einstellung „**Divide by 2 (8MHz input)**“ erlaubt z.B. den Einsatz eines 8 MHz Keramikresonators.

Wird diese Veränderung beim Brennen des Bootloaders vorgenommen, dann gilt sie automatisch auch für die später mit dem Bootloader geladenen Firmware. Das ist dann von Interesse, wenn man einen Quarz einsetzen will, für den es keinen passenden Bootloader gibt (z.B. 12 MHz).

5.4 Nutzbare Interfaces

Die folgende Tabelle zeigt die Zuordnung der einzelnen Pins zu den verfügbaren Interfaces. Man erkennt, das sich die Nutzung einiger Interfaces gegenseitig ausschließt. So kann das LCD1 nicht gleichzeitig mit I2C, SPI, Motor1 oder Motor2 verwendet werden. Man kann aber z.B. gleichzeitig ADC (die 5 EingängeAN0..AN4), Frequenzmesser, RS232, I2C, LCD2, PWM1 und PWM2 verwenden.

Pin	IO-Port	ADC	FRQ	RS 23 2	I2C	SPI	Mwire/ Schieb	LCD 1	LCD 2	PWM	Motor 1..4	L297 1..4	Servo	Counter
2	RA0	AN0									A3			
3	RA1	AN1									B3			
4	RA2	AN2									C3			
5	RA3	AN3									D3			
6	RA4		FRQ											C1
7	RA5	AN4				(SS)								
10	RA6													
21	RB0	AN12			SDA	SDI	SDI	E1			A1	CL1	SB0	
22	RB1	AN10			SCL	SCL	SCL				B1	DIR1	SB1	
23	RB2	AN8				(CS1)		RS	RS		C2	CL2	SB2	
24	RB3	AN9				(CS2)		R/W	R/W		D1	DIR2	SB3	
25	RB4	AN11				(CS3)		D4	D4		A2	CL3	SB4	
26	RB5					(CS4)		D5	D5		B2	DIR3	SB5	
27	RB6					(CS5)		D6	D6		C2	CL4	SB6	
28	RB7					(CS6)		D7	D7		D2	DIR4	SB7	
11	RC0								E2		A4		SC0	C3
12	RC1									PWM2	B4		SC1	
13	RC2									PWM1	C4		SC2	
17	RC6			TX							D4		SC6	
18	RC7			RX		SDO	SDO						SC7	

Den für den USB4all verwendeten Microcontroller PIC18F2455 von Microchip gibt es im 28-Pin DIL-Gehäuse PDIP) und in 28-Pin SMD-ausführung (SOIC). Das folgende Bild zeigt die Pinbelegung.

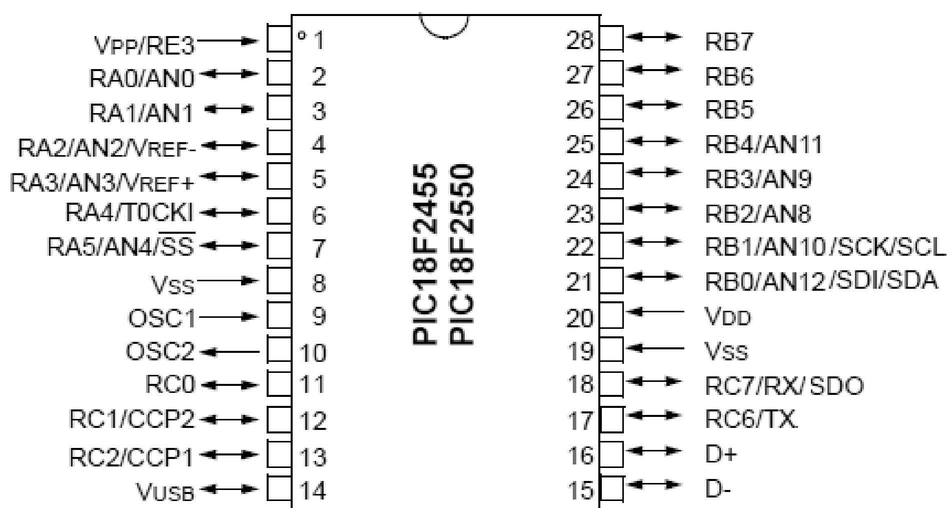


Abbildung 6 Pinbelegung des PIC18F2455

5.4.1 Ausgangspins

Das folgende Bild gilt für alle IO-Pins, auch für RA4 (Beschriftung im Bild ist nicht korrekt).

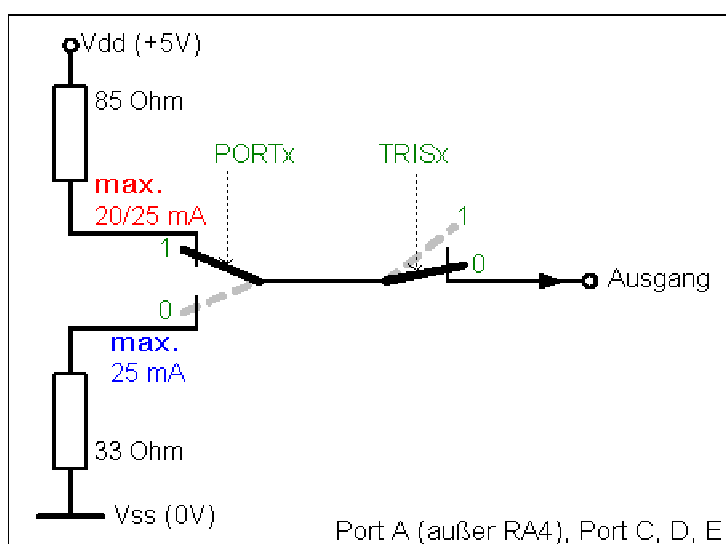


Abbildung 7 Ausgangspin - Prinzip

Für alle Pins gilt, das sie als Ausgangspins einen Strom von jeweils bis zu 20mA bereitstellen können. Die Gesamtlast aller Pins darf aber zusammen 200mA (im USB4all nur 100mA) nicht übersteigen.

Die garantierten Ausgangsspannungspegel sind:

Pegel	allgemein	bei Vdd=5V
High bei 8,5mA Last	> (Vdd-0,7V)	> 4,3V
Low bei -3,5 mA Last	< 0,6V	< 0,6V

Die Ausgangstreiber sind MOSFET-Transistoren mit recht hohen Innenwiderständen. Das

führt dazu, das mit steigendem Ausgangsstrom der Signalpegel immer mehr von Vdd bzw. Vss abweicht.

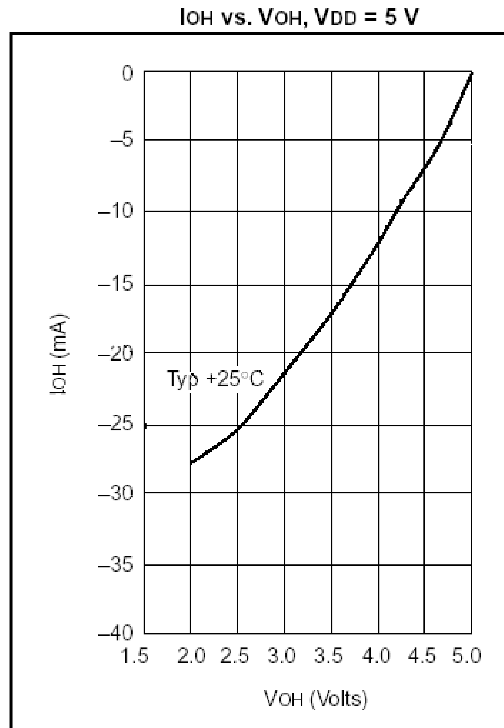


Abbildung 8 High-Pegel bei verschiedenen Ausgangsströmen

Die Grafiken zeigen aber deutlich, dass im Bereich von 0mA bis 20 mA der TTL-Pegel garantiert ist.

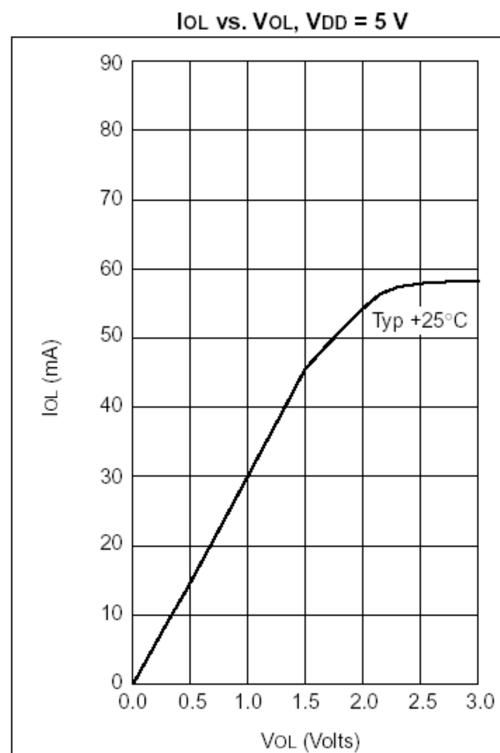


Abbildung 9 Low-Pegel bei verschiedenen Ausgangsströmen

5.4.2 Eingangspins

Als digitale IO-Pins sind (mit Ausnahme von RA4) die Pins der Ports A (RA0..RA3, RA5, RA6) und Port B (RB0..RB7) normale TTL-Eingänge.
Das Pin RA4 sowie die Pins des Port C sind Schmitttrigger-Eingänge (ST). Damit benötigen sie folgende Eingangsspannungspegel:

IO-Input	Funktion	Input-low-Pegel	Input-high-Pegel
RA0..RA3, RA5, RA6, RB0..RB7	TTL	< 0,15 Vdd	>(0.25Vdd+0.8V)
RA4, RC0..RC7	ST	< 0,2 Vdd	> 0,8 Vdd

Bei einem Vdd von typischerweise 5V ergibt das:

IO-Input	Funktion	Input-low-Pegel	Input-high-Pegel
RA0..RA3, RA5, RA6, RB0..RB7	TTL	< 0,75 V	> 2.05 V
RA4, RC0..RC7	ST	< 1 V	> 4 V

Die Stromaufnahme aller Eingänge liegt bei maximal 1 uA.

Für die Pins RB0..RB7 können interne Pull-Up-Widerstände aktiviert werden, die diese Pins mit 50 .. 400 uA nach Vdd ziehen. (Das entspricht Pull-up-Widerständen von ca. 25 kOhm.)

++Achtung++
Eingangsspannungen oberhalb Vdd und unterhalb Vss werden im Chip durch Klemmdioden nach Vdd bzw. Vss abgeleitet. Wenn solche Spannungen an einem Eingang auftreten können, dann ist der Eingangsstrom durch einen Vorwiderstand auf maximal 20mA zu begrenzen.

5.5 TTL-IO-Port-Anschluss

Eine genaue Beschreibung der TTL-IO-Ports und ihrer Verwendung steht unter:
http://www.sprut.de/electronic/pic/grund/ioports.htm#rx

5.6 ADC-Anschluss

Der ADC misst Spannungen zwischen den Referenzspannungen Vss (0V) und Vdd (+5V) mit einer Auflösung von 10 Bit (4.88 mV). Um eine hohe Messgenauigkeit zu garantieren, sollte der Innenwiderstand der Spannungsquelle 2,5kOhm nicht überschreiten.
Spannungen oberhalb Vdd und unterhalb Vss werden durch Klemmdioden abgeleitet. Wenn solche Spannungen am Eingang auftreten können, dann ist der Strom durch einen Vorwiderstand auf maximal 20mA zu begrenzen. Dieser Widerstand verschlechtert natürlich den Innenwiderstand der Spannungsquelle.
Als Referenzspannungen werden standardmäßig Vdd und Vss verwendet, was die Messgenauigkeit aber einschränkt. Alternativ kann man genauere externe Referenzspannungen anschließen. Die dafür nötigen 1..2 Pins verringern dann die Zahl der ADC-Eingänge.
Die positive Referenzspannung wird an AN3 (Pin 5) angeschlossen und die negative referenzspannung an AN2 (Pin 4). Beide Referenzspannungen müssen zwischen Vss und Vdd liegen. Die positive Referenzspannung sollte mindestens 2V höher sein als die negative Referenzspannung.

5.7 Zählfrequenzmesser-Anschluss

Der Zählfrequenzmesser (FRQ, Pin 6) misst Frequenzen bis zu 50 MHz. Der Eingang hat einen Schmittrigger. Der high-Teil des Eingangssignals muss 4V überschreiten (0,8 Vdd), während der low-Teil 1V (0.2 Vdd) unterschreiten muss.
Spannungen oberhalb Vdd und unterhalb Vss werden durch Klemmdioden abgeleitet. Wenn solche Spannungen an einem Eingang auftreten können, dann ist der Strom durch einen Vorwiderstand auf maximal 20mA zu begrenzen.

5.8 RS232-Anschluss

Der RS232-Anschluss sendet und empfängt TTL-Pegel. Diese sind durch einen üblichen RS232-Treiber (z.B. MAX232) in RS232-Pegel zu wandeln (+5V->-12V: 0V->+12V).

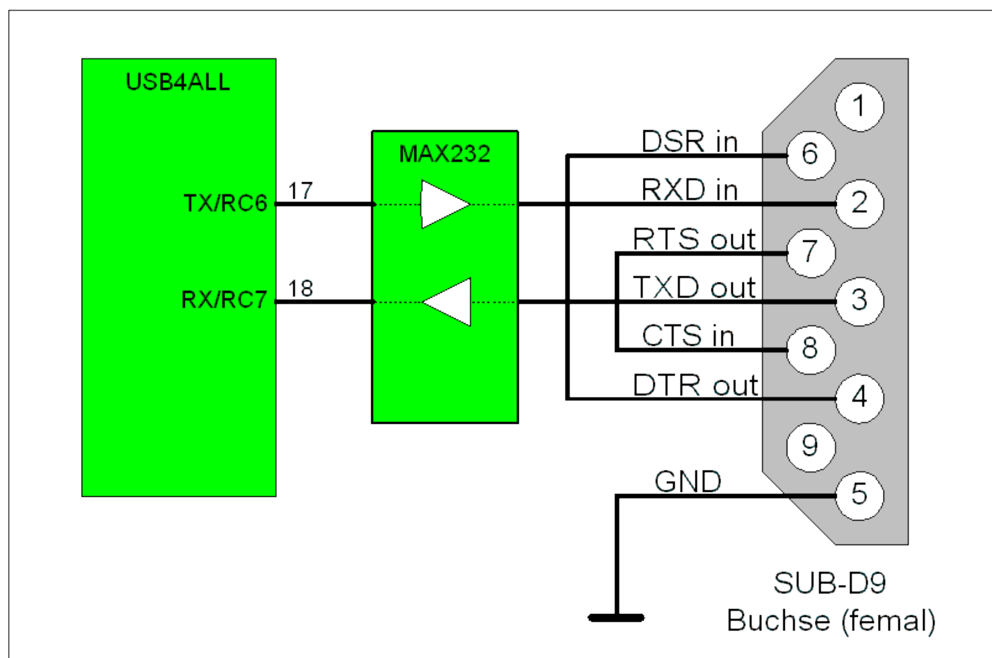


Abbildung 10 RS232-Anschluss

Weitere Informationen findet man auf:
http://www.sprut.de/electronic/interfaces/rs232/rs232.htm
http://www.sprut.de/electronic/pic/grund/rs232.htm

5.9 I2C-Anschluss

Der USB4all kann als Master einen I2C-Bus verwalten. Die beiden Leitungen SDA und SDC benötigen jeweils einen externen pull-up-Widerstand von ca. 1,8 Kilo-Ohm, um den Bus bei Inaktivität auf sicherem High-Pegel zu halten.
Weitere Informationen findet man auf:
http://www.sprut.de/electronic/pic/grund/i2c.htm

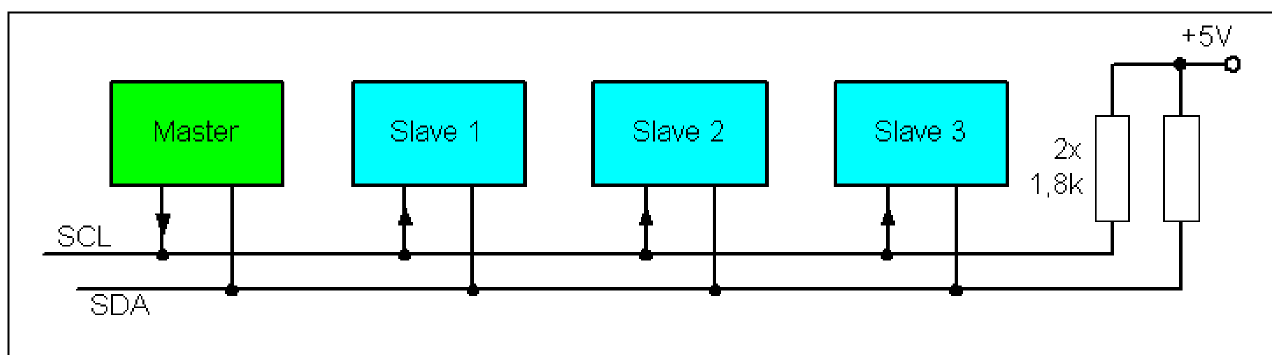


Abbildung 11 I2C-Anschluss

5.10 *LCD-Anschluss*

Der USB4all kann maximal 2 HD44780-kompatible Displays mit bis zu jeweils 2x40 bzw. 4x20 Zeichen ansteuern. Alternativ kann ein Display mit bis zu 4x40 Zeichen angesteuert werden (mit 2 Controllern).

Die LCD-Pins können direkt mit den Pins des USB4all verbunden werden. Die Pins D0...D3 des LCD werden nicht benötigt. Vdd (+5V) und Vss (0V) des Displays sind mit Vdd und Vss des USB4all zu verbinden. Am Pin Vo ist eine (vom Displaytyp abhängige) Kontrastspannung anzulegen.

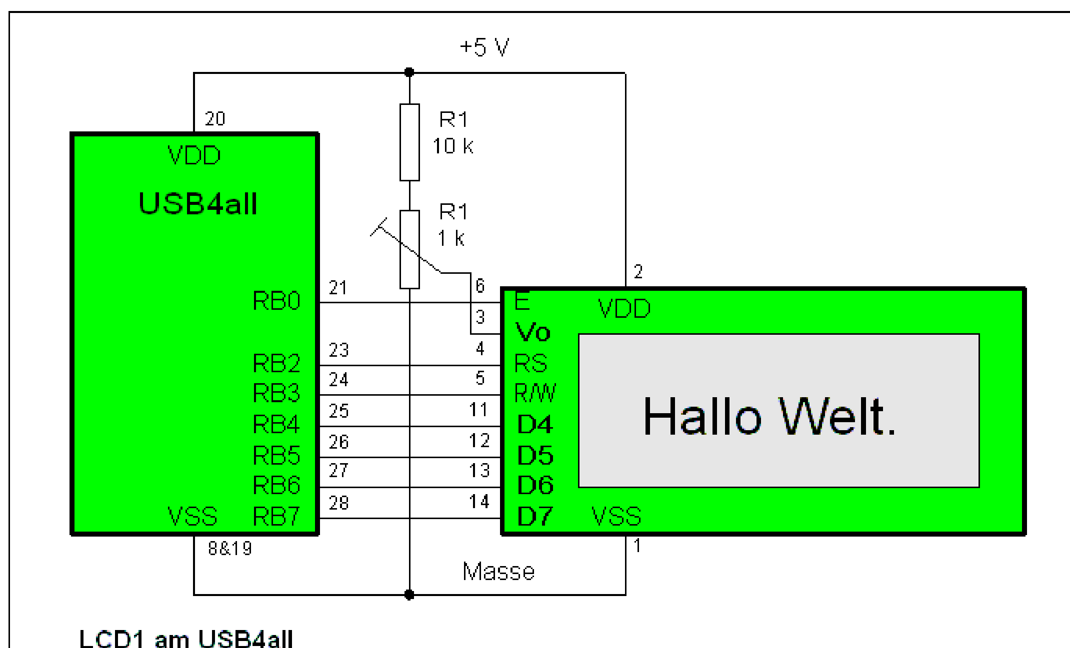


Abbildung 12 LCD-Anschluss

Bei Normaltemperaturdisplays kann die Kontrastspannung durch einen PWM-Kanal mit RC-Siebglied erzeugt werden, ein einfacher Einstellwiderstand tut es aber auch.

Die Schaltung zeigt den Anschluss eines Displays. Soll ein zweites Display angeschlossen werden, so werden dessen Pins mit Ausnahme von “E” parallel zum ersten Display angeschlossen. Der “E”-Anschluss des zweiten Displays kommt an RC0.

Weiter Informationen findet man auf:

http://www.sprut.de/electronic/lcd/index.htm

5.11 PWM-Anschluss

Es gibt zwei PWM-Ausgänge. Beide Pins geben zwar die gleiche Frequenz aus, können aber individuelle Tastverhältnisse haben.

Die Verwendung dieser Pins ist stark von der gewünschten Anwendung abhängig. Soll ein PWM-Kanal als DAC-Ausgang verwendet werden (http://www.sprut.de/electronic/pic/grund/dac/dac.htm#pwm), so ist er mit einem RC-Glied zu sieben. Die Größe des verwendeten Kondensators und des Widerstandes sind von der gewählten PWM-Frequenz abhängig.

5.12 Schrittmotor-Anschluss

5.12.1 Schrittmotoransteuerung mit ABCD-Phasen

Diese Anschlüsse eignen sich zum Anschluss unipolarer Schrittmotoren kleinerer Leistung.
Die pro Kanal 4 Ausgangspins des USB4all müssen mit einem geeigneten Treiber verstärkt werden. Diese können sowohl aus Einzeltransistoren wie auch aus speziellen Treiberschaltkreisen aufgebaut werden. Es ist darauf zu achten, das die Treiber sowohl die Strom- und Spannungsbelastung wie auch die induktive Belastung durch den Motor verkraften.
Dafür gibt es Spezialschaltkreise wie den ULN2075B oder L298. Man kann sowas aber auch schnell selbst zusammenbauen, wie die nachfolgende Schaltung zeigt.

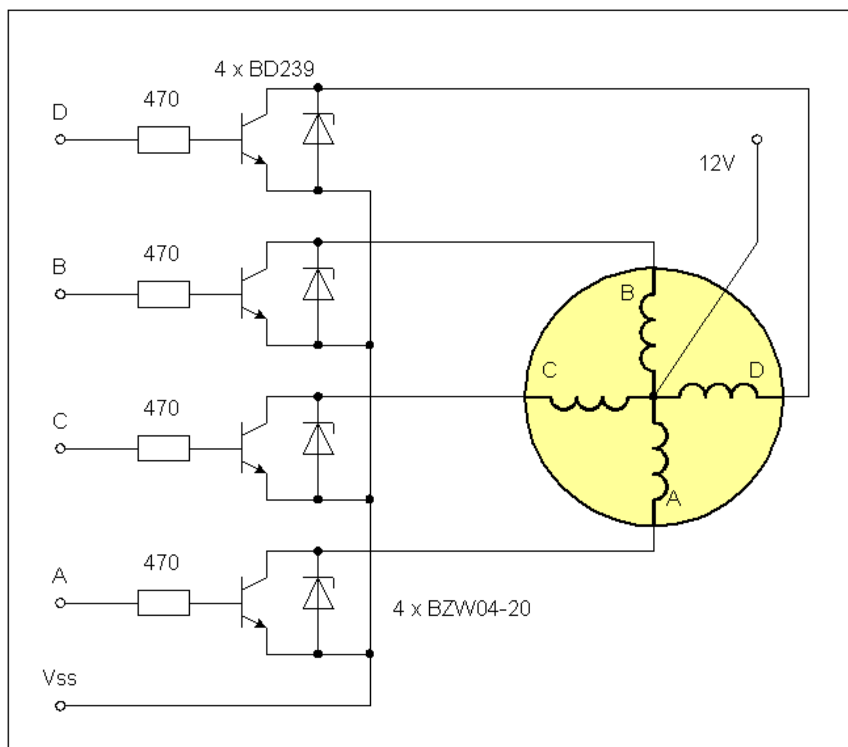


Abbildung 13 einfacher Schrittmotoranschluss

5.12.2 Schrittmotoransteuerung mit L297

Der Schaltkreis L297 bildet zusammen mit dem L298 eine preiswerte und sehr leistungsfähige Lösung zur Schrittmotorensteuerung, die auch den Motorstrom regelt. Aus diesem Grunde unterstützt USB4all diese populären Schaltkreise. Bis zu 4 Kanäle können angesteuert werden.
Die pro Kanal 2 Ausgangspins des USB4all müssen jeweils mit den Pins Clock (CLK) und Direction (CW/CCW) eines Schaltkreises L297 verbunden werden. Das Enable-Pin des L297 ist permanent mit High-Pegel zu verbinden. Der Half/Full-Pin des L297 ist mit dem für die gewünschte Motorbetriebsart nötigen Pegel zu verbinden.

5.13 Servo-Anschluss

Es gibt 13 Servo-Ausgänge zum Anschluss normaler Modellbauservos. Die Ausgangssignale sind positive Pulse mit TTL-Pegel, die direkt in den Eingang eines Servos eingespeist werden können.

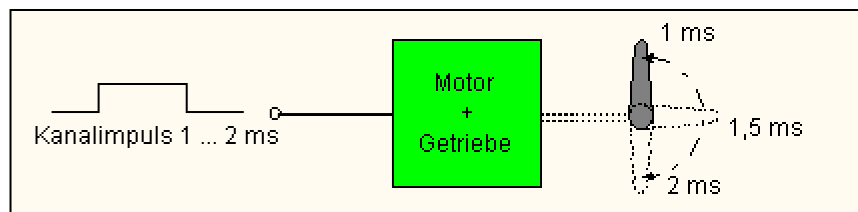


Abbildung 14 Modellbauservo

6 Software

6.1 USB-Device

Das Herzstück des USB4all ist die USB4all-Firmware. Sie könnte mit einem Programmiergerät direkt in einen PIC18F2455 gebrannt werden, dann stünde aber für spätere Updates der Bootloader nicht zur Verfügung.

Es ist besser, zunächst nur den Bootloader-5 in den PIC18F2455 zu brennen. Danach verwendet man meine Windows-Software USBoot (liegt im ZIP-Archiv von USB4all), um die Firmware in den PIC18F2455 nachzuladen. Dann befinden sich Bootloader und Firmware im PIC. Beim Anschließen des USB4all wird nun normalerweise die Firmware gestartet. Möchte man den Bootloader starten (z.B. zum Einspielen einer aktuelleren Firmware), dann ist vor dem Anschließen des USB4all an den PC das Pin 1 mit Vss (Masse / Gnd) zu verbinden. Dazu dient normalerweise der Jumper JP1. Ein Stück Draht tut’s notfalls aber auch.

Die Software USBoot (die für die Nutzung des Bootloaders ohnehin nötig ist) kann den Bootloader des USB4all-MCD (aber nicht des USB4all-CDC) auch ohne gesetzten Jumper starten.

6.2 PC

Es gibt zwei Versionen des USB4all:

Der USB4all-CDC
Unter Windows simuliert der **USB RS-232 Emulation Driver** eine serielle RS232-Schnittstelle (z.B. COM3 oder COM4) auf die mit Windows-üblichen Mitteln zugegriffen werden kann

Der USB4all-MCD
Unter Windows dient der **Microchip Custom Driver** zur Ansteuerung des USB4all über eine spezielle DLL.

6.2.1 Treiberinstallation

Nachdem das Gerät mit dem USB4all fertig aufgebaut wurde, und der Steuer-PIC die korrekte Firmware oder wenigstens den Bootloader eingebrannt bekam, muss er nun noch unter Windows eingerichtet werden.

Der USB4all-MCD benötigt den **Microchip Custom Driver** (mpusbapi.dll). Den findet man auf der Microchip Homepage, oder im Softwarepaket zum USB4all auf meiner Homepage. Der gleiche Treiber wird auch für meinen Brenner8 und meinen Bootloader verwendet. Wer den Treiber z.B. für den Brenner8 schon mal eingerichtet hat, kann sich die Installation sparen. Wer den Treiber bisher nur für ein Microchip-USB-Testboard eingerichtet hat, der muss ihn allerdings noch einmal installieren. Momentan ist es noch nicht zulässig den Brenner8 und USB4all-MCD gleichzeitig an einen PC anzuschließen

Der USB4all-CDC benötigt den **USB RS-232 Emulation Driver**. Der ist Bestandteil von Windows. Fragt Windows nach den Treiber für das neue USB-Gerät, dann wählt man einfach manuell die mit der Firmware mitgelieferte INF-Datei aus. Das USB4all-CDC

erscheint danach als COM3- oder COM4-Port im Gerätemanager.
Der zum USB4all gehörende Bootloader-5 benötigt aber auch den **Microchip Custom Driver**. Die Installation ist aber kein Problem.

6.2.2 Installation: Microchip Custom Driver

Die nachfolgenden Bilder stammen von einer den **Microchip Custom Driver**-Installation für den Brenner8, die aber mit einer USB4all-Installation (bis auf den Namen) identisch ist.

Man lädt man sich das USB4all-Zip-File von meiner Homepage,

http://www.sprut.de/electronic/soft/usburn/usburn.htm - download

und entpackt es in einen Ordner auf der lokalen Festplatte.
Der Treiber liegt dann im Unterordner **driver** .

Nun kann das Gerät mit dem USB4all-MCD oder Bootloader-5 an den Windows-PC angesteckt werden. Windows findet automatisch das ihm noch unbekannte USB-Gerät **sprut-device**.

Abbildung 15 Neue Hardware gefunden

Anschließend fordert Windows zur Treiberinstallation auf.

Abbildung 16 Hardware Assistent 1

Nach einem Klick auf **Weiter** ist man im Assistenten für die Hardwareinstallation.

Abbildung 17 Hardware Assistent 2

Hier wählt man die untere der beiden möglichen Optionen aus, um den Treiber manuell auswählen zu können. Danach klickt man auf **Weiter**.

Nun muss man das Gerät in eine Geräteklasse einordnen. Da es nicht so richtig in eine der üblichen Gruppen passt, wählt man **Andere Geräte**, und klickt auf **Weiter**.

Abbildung 18 Hardware Assistent 3

Nun kommt man zur Auswahl des Gerätetreibers. Hier klickt man auf die Schaltfläche **Datenträger...**

Abbildung 19 Gerätetreiber auswählen 1

Im folgenden Fenster muss nun der Pfad zum Verzeichnis mit dem Treiber eingestellt werden.

Abbildung 20 Pfad zum Treiber einstellen

Windows schaut in diesem Verzeichnis nach, und finden einen passenden Treiber für ein **sprut-device**.

Abbildung 21 Gerätetreiber auswählen 2

Im folgenden Fenster wird die Auswahl mit **Weiter** bestätigt.

Abbildung 22 Treiber kann installiert werden

Windows installiert nun den Treiber. Mit einem Klick auf **Fertig stellen** wird die Installation abgeschlossen.

Abbildung 23 Assistent fertig stellen

Von nun an findet man das Gerät im Gerätemanager.

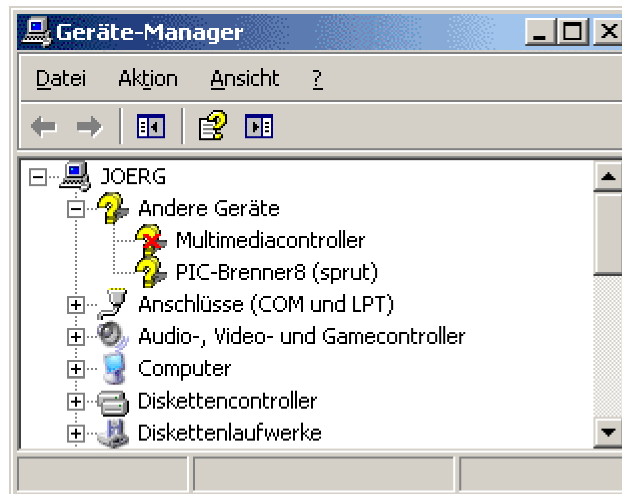


Abbildung 24 Gerätemanager

Im Gerätemanager sollte dem Betriebssystem verboten werden, das Gerät abzuschalten um Energie zu sparen. Ansonsten kann es passieren, das es extrem langsam arbeitet.

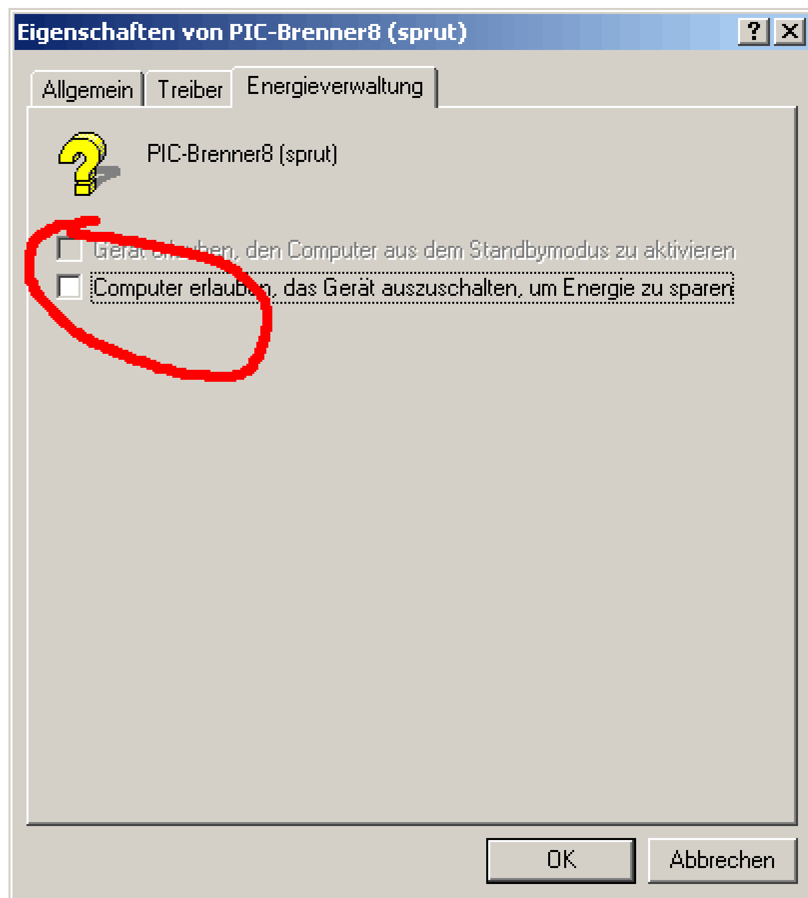


Abbildung 25 Gerätemanager - Energiesparoption

Damit ist das Gerät mit dem USB4all einsatzbereit.

6.2.3 Installation: USB RS-232 Emulation Driver

Um unter Windows den RS232 Emulation Driver zu installieren benötigt man lediglich die im USB4all-ZIP-File mitgelieferte Datei **mchpcdc.inf**.
Nach dem Anstecken des USB4all-CDC an den PC fordert Windows zur Treiberinstallation auf: Man untersagt Windows, Windows-Update zu kontaktieren.

Abbildung 26 Beginn der Treiberinstallation

Stattdessen wählt man aus, den Treiber manuell auszusuchen

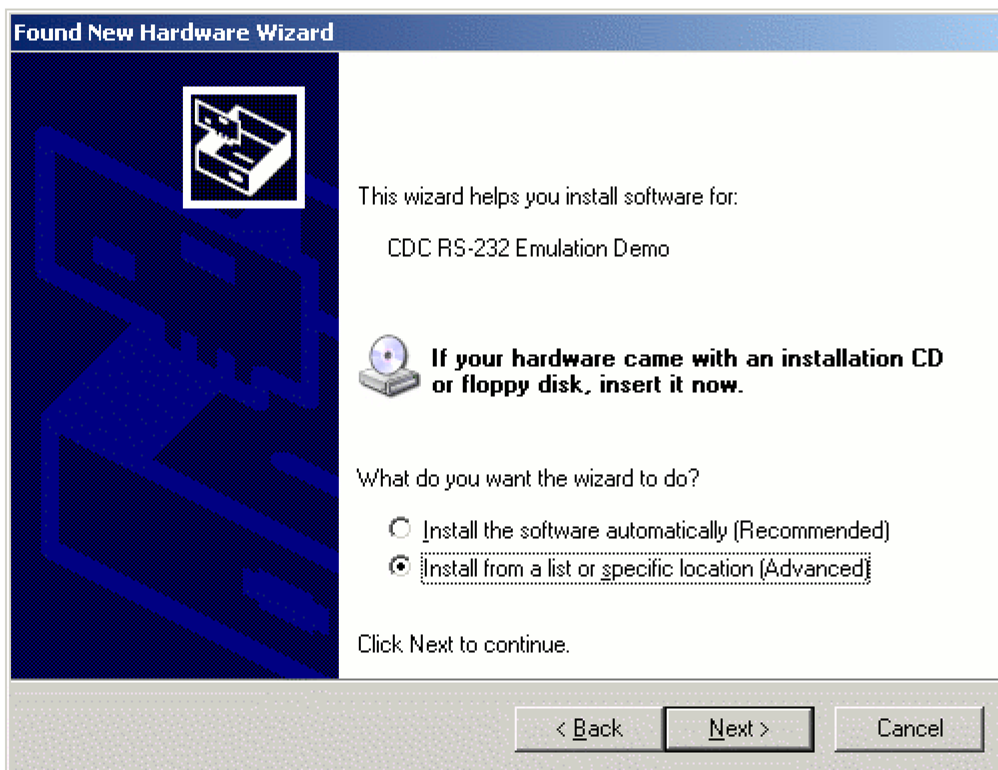


Abbildung 27 manuelle Auswahl des Treiber benutzen

Im folgenden Fenster wird der Pfad zur mchpcdc.inf-Datei angegeben.

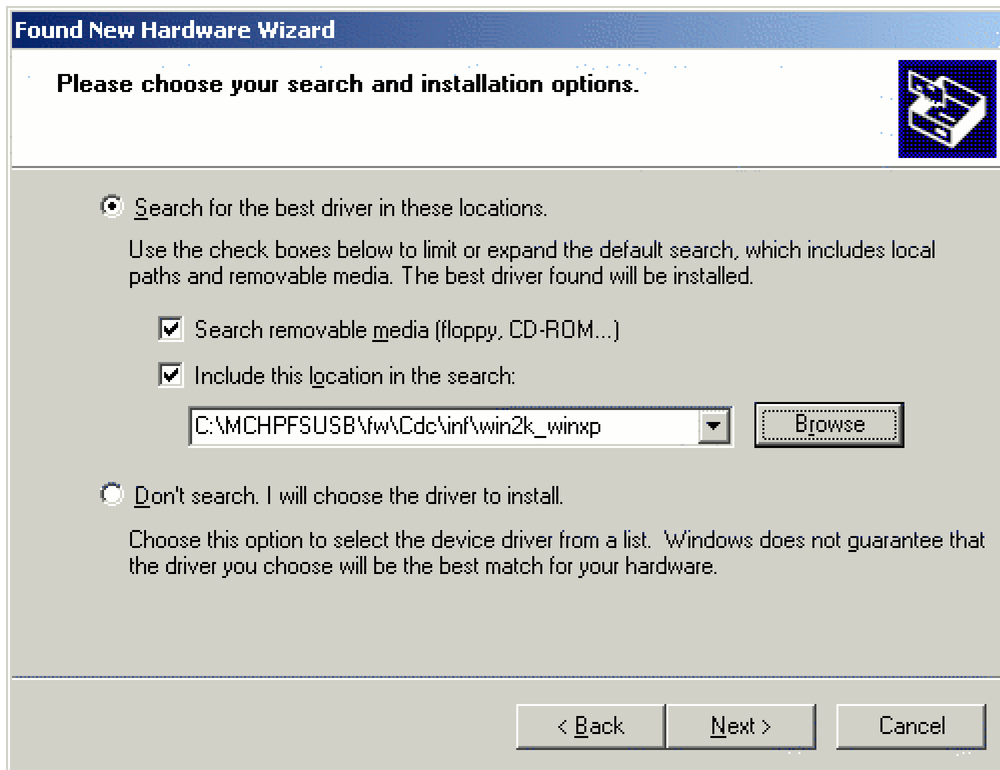


Abbildung 28 den Pfad zur inf-Datei angeben

Die anschließende Warnung übergeht man mit „Weiter“ bzw. „Continue“.

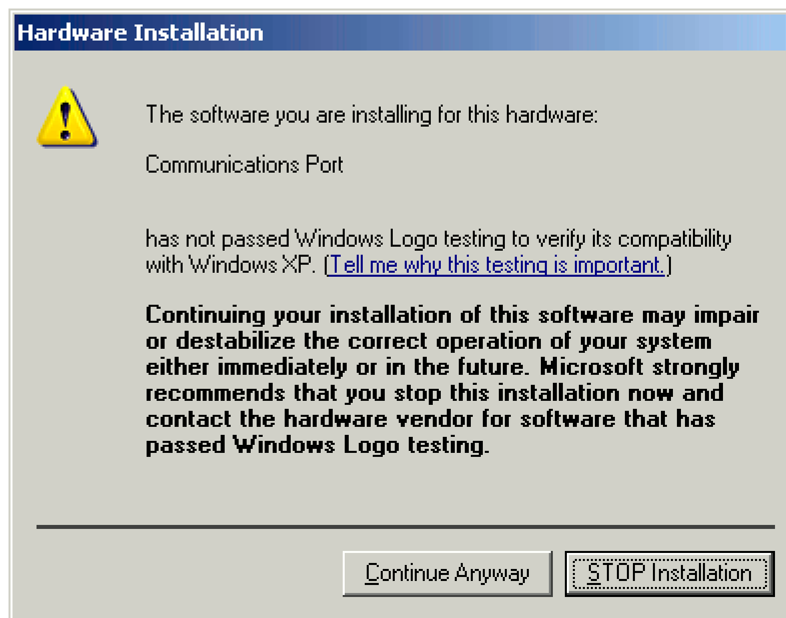


Abbildung 29 den Windows-Logo-Test ignorieren

Und schon ist man fertig.

Abbildung 30 Ende der Installation

Der USB4all-CDC ist im Gerätemanager jetzt als COM3 oder COM4 zu finden.

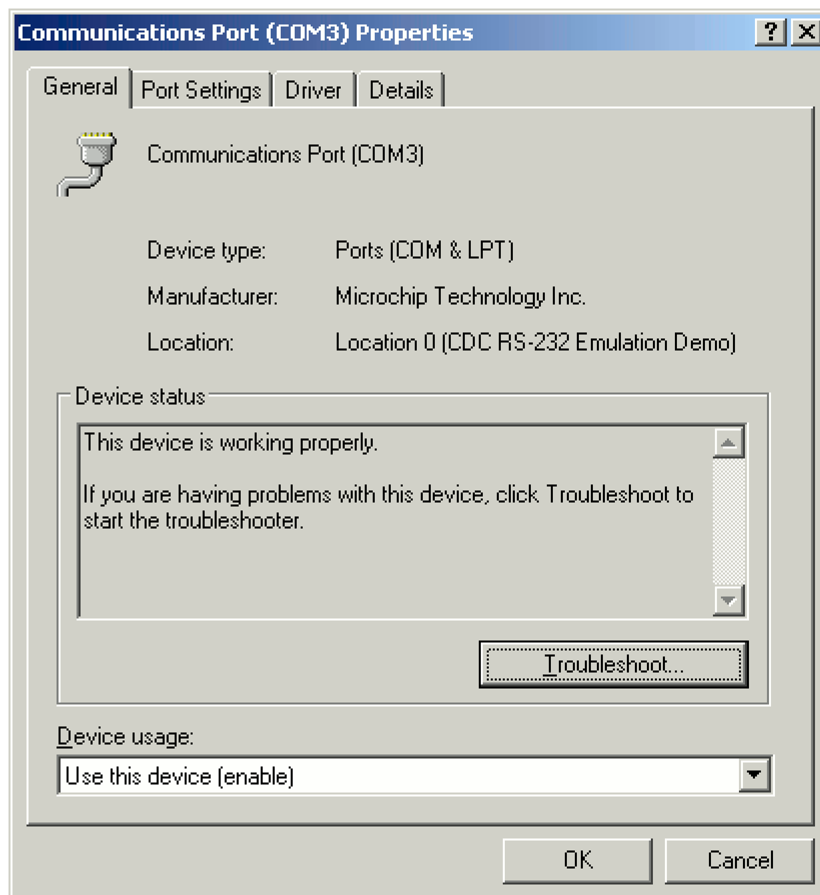


Abbildung 31 USB4all-CDC als emuliertes COM3-Port

7 Befehlsübersicht

Jeder Befehl besteht aus einer kurzen Kette von Bytes (einem String). Dieser String wird zum USB4all gesendet. Der USB4all arbeitet den Befehl ab, und sendet dann als Quittung einen 16 Byte langen String zurück.
Bei Schreiboperationen oder Initialisierungsbefehlen dient dieser String nur als Lebenszeichen des USB4all. Echte Daten enthält er nur nach Leseoperationen.

Die Länge des an den USB4all zu sendenden Strings darf 64 Bytes nicht überschreiten. Die meisten Befehle kommen aber mit 2 .. 4 Byte aus.

Beim USB4all-CDC wird der Byte-String in einen ASCII-String "eingebettet". Dieser ist 3 mal so lang, wie der Byte-String. Die Länge ASCII-Strings darf 64 Zeichen nicht überschreiten, was die Zahl der enthaltenen Daten-Bytes auf 20 begrenzt.

Das erste Byte des Befehlsstrings adressiert das Subsystem des USB4all, an das der Befehl geschickt werden soll. Das kann z.B. der ADC oder das RS232-Interface sein. Folgende Subsysteme sind definiert, die kursiv-durchgestrichen gedruckten sind noch nicht realisiert:

- 0x50 TTL-IO-Pins
- 0x51 10-Bit-ADC
- 0x52 Frequenzmesser
- 0x53 SPI-Interface
- 0x54 I2C-Interface
- 0x55 Dotmatrix-LCD-Display Nr. 1
- 0x56 Dotmatrix-LCD-Display Nr. 2
- 0x57 PWM-Ausgang 1
- 0x58 PWM-Ausgang 2
- 0x59 RS232-Interface
- 0x5A interner EEPROM
- ~~*0x5B Tastenmatrix*~~
- 0x5C Schrittmotor Nr. 4
- 0x5D Schrittmotor Nr. 1
- 0x5E Schrittmotor Nr. 2
- 0x5F Schrittmotor Nr. 3
- 0x60 L297 Schrittmotorsteuerung Nr. 1
- 0x61 L297 Schrittmotorsteuerung Nr. 2
- 0x62 L297 Schrittmotorsteuerung Nr. 3
- 0x63 L297 Schrittmotorsteuerung Nr. 4
- 0x64 Servosteuerung B
- 0x65 Servosteuerung C
- 0x66 Schieberegister
- 0x67 Mikrowire
- 0x68 Counter 0
- 0x69 Counter 3
- 0xFF Reset

Das zweite Byte ist der eigentliche Befehl. Er folgt (soweit bei den Subsystemen sinnvoll)

folgendem Schema:

- 0x00 Abschalten des Subsystems
- 0x01 Initialisieren des Subsystems
- 0x02 Senden eines Bytes
- 0x03 Lesen eines Bytes
- 0x04 Senden eines Strings
- 0x05 Lesen eines Strings

Ab dem 3. Byte folgen dann (wenn nötig) Daten für den jeweiligen Befehl.

7.1 TTL-IO-Pins

Elektrische Spezifikation
Der USB4all hat 20 IO-Pins, die als Eingang oder Ausgang für TTL-Pegel dienen können. Ein auf “Ausgang” geschaltetes Pin kann bis zu 25mA abgeben. Die Summe aller Ausgangsstöme darf aber 200mA nicht überschreiten. (Da dem USB4all nur maximal 100mA aus dem USB-Bus zur Verfügung stehen, ist das Limit 100mA.)

Die als Eingang programmierten Pins sind mit Klemmdioden nach Vss (Masse) und Vdd (+5V) abgesichert. Deshalb dürfen sie nur mit Pegeln zwischen Vss und Vdd direkt verbunden werden. Bei höheren oder niedrigeren Eingangsspannungen sind Reihenwiderstände einzusetzen , die den Klemmstrom auf unter 20mA begrenzen.
Die Verbindung mit eine +12V-Quelle erfordert einen Widerstand von mindestens 350 Ohm. Ein Wert von 22 kOhm wäre angemessen.

Nach dem Einschalten der Betriebsspannung sind alle Pins als digitale Eingänge konfiguriert.

Die 20 IO-Pins sind in 3 Ports angeordnet:
- PortA mit den 7 Pins RA0..RA6
- PortB mit den 8 Pins RB0..RB7
- PortC mit den 5 Pins RC0..RC2 und RC6..RC7

Benötigen andere Subsysteme Pins für ihre Funktion, so fehlen diese als TTL-IO-Pins. Das RS232-Interface benötigt z.B. die Pins RC6 und RC7. Wird das RS232-Subsystem initialisiert, dann nimmt es sich diese beiden Pins. Wird das RS232-Subsystem wieder abgeschaltet, dann gibt es die beiden Pins wieder zurück.

Es gibt 8 unterschiedliche Befehle für IO-Pins.

Byte0 Subsystem	**Byte1 Befehl**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**
0x50 IO-Pins	0x00 aus alle Output-Pins abschalten	-	-	-	-
	0x01 initialisieren	In/out-Maske für PortA	In/out-Maske für PortB	In/out-Maske für PortC	Pull-up für PortB: 0: aus 1: an
	0x02 Ausgabe auf alle Ports	Wert für PortA	Wert für PortB	Wert für PortC	-
	0x03 Lesen von allen Ports	-	-	-	-
	0x04 einzelne Pins auf Input schalten	In-Maske für PortA	In-Maske für PortB	In-Maske für PortC	-
	0x05 einzelne Pins auf Output schalten	Out-Maske für PortA	Out-Maske für PortB	Out-Maske für PortC	-
	0x06 einzelne Pins auf 5V setzen	High-Maske für PortA	High-Maske für PortB	High-Maske für PortC	-
	0x07 einzelne Pins auf 0V	Low-Maske für PortA	Low-Maske für PortB	Low-Maske für PortC	-

	setzen				

USB4all antwortet auf alle Befehle mit 16-Byte. Nur beim Befehl 3 enthalten diese Bytes eine Information:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
0x50	0x03	Wert von PortA	Wert von PortB	Wert von PortC	-

Die Steuerung der 20 TTL-IO-Pins erfolgt über drei Bytes (MaskeA, MaskeB, MaskeC). Jedes IO-Pin ist einem Bit in einem Byte in einer dieser Masken zugeordnet.

Maske	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MaskeA	-	RA6	RA5	RA4	RA3	RA2	RA1	RA0
MaskeB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
MaskeC	RC7	-	-	RC4	-	RC2	RC2	RC0

0x03-Lesen von allen Ports
Wird mit dem Befehl 0x03 der Wert aller Pin gelesen, so werden an den PC die drei Bytes (Masken) übermittelt. Jedem Pin ist genau ein Bit in einer Maske zugeordnet (siehe oben). Ein Low-Pegel an einem Pin entspricht einer “0” in der Maske. Ein High-Pegel entspricht einer “1”.

0x01-initialisieren
Das TTL-IO_Pin-Subsystem funktioniert eigentlich auch schon ohne Initialisierung. Man kann sofort mit dem Befehl 0x03 die Pegel an allen Pin auslesen. Allerdings sind alle Pins als Input-Pins definiert. Es lassen sich deshalb noch keine TTL-Spannungen ausgeben (z.B. an eine LED).
Um mit einem Pin einen TTL-Pegel auszugeben, muss dieses Pin zunächst zu einem Output-Pin umprogrammiert werden. Das erfolgt mit dem Befehl 0x01.

Dem USB4all werden drei Masken übergeben. In denen ist für jedes Pin festgelegt, ob es sich um ein Input-Pin (1) oder ein Output-Pin (0) handeln soll. Nach den Masken folgt noch ein Byte. Hat dieses “Byte 5” den Wert ”1”, dann werden für alle Input-Pins des PortB interne pull-up-Widerstände aktiviert.

Der String (0x50-0x01-0x00-0x00-0x00-0x00) macht alle Pins zu Ausgängen.
Der String (0x50-0x01-0xFF-0xFF-0xFF-0x00) macht alle Pins wieder zu Eingängen.

0x04-einzelne Pins auf Input schalten
0x05-einzelne Pins auf Output schalten
Der Befehl 0x01 wirkt immer auf alle Pins. Will man nur einzelne Pins auf input oder output schalten, dann benutzt man die Befehle 0x04 und 0x05.
Hier werden ebenfalls Masken für alle drei Ports übermittelt, die Bedeutung ist aber eine andere.
Eine “0” in der Maske bedeutet, das die Funktion des Pins nicht verändert wird. Eine “1” bedeutet dagegen, dass das betroffene Pin auf Input (Befehl 0x04) bzw. auf Output (Befehl 0x05) geschaltet wird.

Der String (0x50-0x04-0x01-0x80-0x00) macht die Pins RA0 & RB7 zu Eingängen.
Der String (0x50-0x05-0x01-0x00-0x00) macht das Pin RA0 zu einem Ausgang.

0x02-Ausgabe auf alle Ports

Mit diesem Befehl lassen sich alle auf Output programmierten Pins gleichzeitig auf einen individuellen TTL-Pegel setzen. Dazu werden die drei Masken an den USB4all übermittelt. Eine “0” in einer Maske entspricht dem Low Pegel (0V), eine “1” dem High-Pegel (5V).

Input-Pins reagieren natürlich nicht, sie merken sich die Werte aber. Werden sie später auf Output umgeschaltet, dann beginnen sie sofort, den gemerkten Pegel auszugeben.

0x06-einzelne Pins auf +5V setzen
0x07-einzelne Pins auf 0V setzen

Der Befehl 0x02 wirkt immer auf alle Output-Pins. Will man nur einzelne Pins auf high oder low setzen, dann benutzt man die Befehle 0x06 und 0x07.
Hier werden ebenfalls Masken für alle drei Ports übermittelt, die Bedeutung ist aber eine andere.
Eine “0” in der Maske bedeutet, das des Pegel des Pins nicht verändert wird. Eine “1” bedeutet dagegen, dass das betroffene Pin auf High (Befehl 0x06) bzw. auf Low (Befehl 0x07) geschaltet wird.

Der String (0x50-0x06-0x01-0x80-0x00) schaltet die Pins RA0 & RB7 auf High.
Der String (0x50-0x07-0x01-0x00-0x00) schaltet nur das Pin RA0 wieder auf Low.

7.2 10-Bit-ADC

Der USB4all hat 10 umschaltbare Eingänge für den 10-Bit ADC. Mit ihnen können Spannungen zwischen 0V und 5V mit einer Auflösung von ca. 4,9 mV gemessen werden. Die 10 ADC-Eingänge belegen Pins des PortA und des PortB:

- AN0 = RA0
- AN1 = RA1
- AN2 = RA2
- AN3 = RA3
- AN4 = RA5 (!)
- AN8 = RB2
- AN9 = RB3
- AN10 = RB1
- AN11 = RB4
- AN12 = RB0

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um den ADC zu nutzen, muss dieser mit dem Befehl 0x01 aktiviert werden. Dabei werden eine vom Nutzer gewählte Zahl von ADC-Eingängen dem digitalen IO-Pins „weggenommen“. Solange diese Pins dem ADC zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des ADC (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

Es gibt 4 unterschiedliche Befehle für den ADC.

Byte 0 Subsystem	**Byte1 Befehl**	**Byte 2**	**Byte 3**
0x51 10-Bit-ADC	0x00 ausschalten, alle ADC-Pins sind wieder digital	-	-
	0x01 initialisieren	Anzahl der ADC-Eingänge (0..10) 0: gar keiner 1: AN0 2: AN0..AN1 3: AN0..AN2 4: AN0..AN3 5: AN0..AN4 6: AN0..AN4,AN8 7: AN0..AN4,AN8..9 8: AN0..AN4, AN8..10 9: AN0..AN4, AN8..11 10: AN0..AN4, AN8..12	Referenzspannung: 0: Vss=Vref-/Vdd=Vref+ 1: Vss=Vref-/AN3=Vref+ 2: AN2=Vref-/Vdd=Vref+ 3: AN2=Vref-/AN3=Vref+
	0x02 festlegen des für die nächste Messung verwendeten ADC-Eingangs	0: AN0 1: AN1 2: AN2 3: AN3 4: AN4 5: AN8 6: AN9 7: AN10 8: AN11 9: AN12	-
	0x03 Messen der Spannung	-	-

USB4all antwortet auf alle Befehle mit 16-Byte. Nur beim Befehl 3 enthalten diese Bytes eine Information:

Byte 0	Byte 1	Byte 2	Byte 3
0x51	0x03	Low	High

Das ADC-Messergebnis ist ein 10-Bit-Wert. Low enthält die unteren 8 Bit, während High die oberen 2 Bit enthält.

Referenzspannungen

Alle Messungen bezieht der ADC auf den Spannungsbereich zwischen der minimalen Referenzspannung Vref- und der maximalen Referenzspannung Vref+. Eine Eingangsspannung, die so hoch ist wie Vref-, wird in den Messwert 0 gewandelt. Eine Spannung von Vref+ ergibt 1023.

Im einfachsten Fall verwendet man Vss als Vref- und Vdd als Vref+. Bei einer Betriebsspannung von genau 5V (Vdd) misst der ADC dann Spannungen zwischen 0V und 5V mit einer Auflösung von 4,88mV (5V/1024). Diese Einstellung erreicht man, indem man beim Befehl 0x01 als Byte 3 den Wert 0x00 übermittelt. Der Chip verbindet dann Vss und Vdd chipintern mit Verf- und Vref+.

Der String (0x51 - 0x01 - 0x01 - 0x00) initialisiert den ADC für die Nutzung von AN0.

Der String (0x51 - 0x02 - 0x00) wählt den Eingang AN0 für die nächste Messung aus.

Der String (0x51 - 0x03) misst nun die Spannung an AN0.

Die Antwort des USB4all auf den letzten String könnte sein (0x51 - 0x03 - 0x12 - 0x03 - ..).

Das Messergebnis wäre die hexadezimale Zahl 0x0312, was im dezimalen System 786 bedeutet. Da als Referenzspannung Vss (0V) und Vdd (5V) dienten, entspricht das

5V / 1023 * 786 = 3,84V.

In der Regel ist die Betriebsspannung einer digitalen Schaltung aber weder genau noch stabil genug, um die 10-Bit-Auflösung des ADC voll nutzen zu können.

Für genaue Messungen benötigt man genaue und stabile Referenzspannungen. Die lassen sich dem USB4all über die Pins AN2 und AN3 von einer externen Referenzspannungsquelle zuführen. Natürlich entfallen dann diese Pins als Messeingänge. Die entsprechende Einstellung nimmt man im Byte 3 des Befehls 0x01 vor. Die externen Referenszspannungen dürfen weder unter Vss-Pegel noch über Vdd-Pegel liegen. Außerdem muß Vref- immer kleiner als Vref+ sein. Die Differenz zwischen beiden Referenzspannungen sollte nicht kleiner als 2V sein, um eine hohe Meßgenauigkeit zu garantieren.

Ein guter Kompromiss ist es oft, Vss als negative Referenzspannung Vref- zu belassen, und nur eine stabile positive Referenzspannung Vref+ von 2,5 .. 5V via AN3 anzuschließen. In diesem Fall kann AN2 weiterhin als analoger Meßeingang dienen (Byte3 = 0x01).

+HINWEIS+

Wenn man im laufenden Betrieb die Anzahl der verwendeten ADC-Eingänge ändern will (Befehl 0x01), sollte man voher den ADC abschalten (Befehl 0x00).

7.3 Zählfrequenzmesser

Der USB4all hat einen Eingang zur Messung einer Frequenz. Mit ihm können Frequenzen von 150 kHz bis 50 MHz mit einer Genauigkeit von etwa 5 Dezimalstellen (Fehler < 0,013%) gemessen werden. Bei niedrigeren Frequenzen beträgt der Messfehler nicht mehr als 10 Hz .. 20 Hz..
Der Eingang hat einen Schmitttrigger. Der high-Teil des Eingangssignals muss 4V überschreiten, während der low-Teil 1V unterschreiten muss.

Der FRQ-Eingang belegt ein Pin des PortA:

- FRQ = RA4

Der Frequenzmesser funktioniert ohne Initialisierung. Das Pin RA4 wird vom Frequenzmesser auf Input geschaltet (wenn es nicht schon vorher auf Input gestellt war). Ein Zurückstellen auf Output (falls es vorher Output war) erfolgt nach der Frequenzmessung aber nicht.

Es gibt 2 unterschiedliche Befehle für den Frequenzzähler:

- Beim Befehl **0x05** wählt der USB4all selbständig den besten Vorteiler, und misst die Frequenz mit bester Genauigkeit (Messzeit 100ms). Als Ergebnis wird die Frequenz in Herz als 32-Bit-Wert zum PC übermittelt.
- Beim Befehl **0x03** kann man den Vorteiler und die Messzeit “manuell” einstellen. Das Ergebnis ist der 16-Bit Zählwert, aus dem man dann die Frequenz erst noch ausrechnen muss.

++ACHTUNG++
Der Zählfrequenzmesser unterbricht alle anderen laufenden Prozesse für die Dauer der Messung (bis zu 100ms lang). Gleichzeitig asynchron drehende Schrittmotoren werden also unterbrochen, laufen nach der Frequenzmessung aber weiter.

Byte 0 **Subsystem**	**Byte1** **Befehl**	**Byte 2**	**Byte 3**
0x52 FRQ	0x03 Frequenzmessung manuell	Interner Vorteiler: 8: 1:1 0: 2:1 1: 4:1 2: 8:1 3: 16:1 4: 32:1 5: 64:1 6: 128:1 7: 256:1	Messzeit: 0: 10ms 1: 1ms 2: 10ms 3: 100ms
0x52 FRQ	0x05 Frequenzmessung autorange	-	-

USB4all antwortet auf einen Befehl mit 16-Byte, die nach dem Befehl 0x03 das Zählergebnis und nach dem Befehl 0x05 die Frequenz beinhalten:

Byte 0	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**
0x52 : ok 0xFF: Überlauf	0x03	Low	High	-	-
0x52 : ok 0xFF: Überlauf	0x05	0. Byte (Low-Bye)	1. Byte	2. Byte	3. Byte (High-Byte)

Das Zählergebnis (Befehl 0x03) ist ein 16-Bit-Wert. Low enthält die unteren 8 Bit, während High die oberen 8 Bit enthält.
Der Frequenzwert (Befehl 0x05) ist ein 32-Bit-Wert. Low-Byte enthält die untersten 8 Bit, während High-Byte die obersten 8 Bit enthält. Die übrigen beiden Byte liegen logischerweise dazwischen.

Die Nutzung der **autorange**-Methode (0x52-0x05) ist denkbar einfach:
Der String (0x52 - 0x05) startet die automatische Messung.
Die Antwort ist z.B. der String (0x52 - 0x05 - 0x40 - 0x78 - 0x7D - 0x01 - ...).
Das Messergebnis ist die hexadezimale Zahl 0x017D7840 was im dezimalen System 25000000 bedeutet. Es wurde also genau 25 MHz gemessen.

Wer die **manuelle** Methode (0x52-0x03) bevorzugt (warum auch immer) muss sich einige zusätzliche Gedanken machen:

Messverfahren
Die am Pin 4 (RA4) anliegende Impulse werden zunächst in einem Vorteiler (mit einstellbarem Teilverhältnis) geteilt. Die dadurch verringerte Frequenz wird einem 16-Bit-Zähler zugeführt. Dieser Zählt die Pulse während einer festgelegten Messzeit.
Sowohl das Vorteiler-Verhältnis wie auch die Messzeit wird im Befehl 0x03 festgelegt.

Vorteiler
Der interne 16-Bit-Zähler des USB4all kann nur Frequenzen bis zu 6 MHz verarbeiten. Um höhere Frequenzen zu zählen, wird der interne Vorteiler benötigt. Dessen Eingang verkraftet 50 MHz (symmetrisch). Werden unsymmetrische Signale eingespeist, dann muss deren High- und Low-Zeit jeweils wenigstens 10 ns betragen.

Messzeit
Der Zähler zählt die während 1 ms, 10 ms oder 100 ms eintreffenden Impulse. Dabei kann er bis maximal 65535 zählen. Treffen in der Messzeit mehr Impulse ein, dann läuft der Zähler über, und in der Quittung wird das Byte 0 auf den Wert 0xFF gesetzt.

Berechnung der Frequenz
Beim Befehl 0x03 liefert der Frequenzzähler nicht direkt die Frequenz, sondern nur das Zählergebnis. Daraus muss der Nutzer im PC noch die Frequenz errechnen. Dazu wird das Zählergebnis zuerst mit dem Vorteilerverhältnis multipliziert, und danach durch die Messzeit (in Sekunden!) dividiert.

<u>Beispiel:</u>
Vorteiler 4:1 / Messzeit 10ms / Zählergebnis 4587
F = 4587 x 4 / 0.01 = 4587 x 4 x 100 = 1,8348 MHz

Beim Befehl 0x05 erledigt der USB4all diese Berechnung selbst.

Genauigkeit
Die Genauigkeit wird durch mehrere Faktoren bestimmt:

- die Taktquelle des USB4all (< 0,005%)
- Zählerauflösung (< 0,003%)
- Firmwaretiming (< 0,0015%)
- die gewählte Messzeit und das Vorteilerverhältnis (<0,003%)

Nach Murphy addieren sich Toleranzen und Fehler immer nur nach einer Seite auf, aber auch dann liegt der Messfehler noch unter 0,013%. Da entspricht immerhin 5 Stellen einer Dezimalzahl.

Taktquelle:
Ein Quarz aus der Massenproduktion hat (ohne Trimmung) einen Frequenzfehler von ca. 0,005%. Dieser Fehler geht direkt in das Messergebnis ein. Ein Keramikresonator hat einen Fehler von ca. 0,5%. Wenn man den Frequenzmesser benutzen möchte, sollte man besser einen Quarz verwenden.

Zählerauflösung:
Da ich nur einen 16 Bit langen Zähler verwende, ist die Auflösung des Zählers nie besser als 0,0015% des Skalenendwertes. Durch den richtigen Einsatz der Vorteiler lässt sich dieser Auflösungsfehler immer unter 0,003% vom Messwert halten.

Firmwaretiming:
Die Messzeit wird in der Firmware durch eine Warteschleife erzeugt, und ist bei1 ms und 10 ms Messzeit eine Kleinigkeit zu lang. Das führt bei 10ms Messzeit zu einem Messfehler von ca. +0,03%, bei 1ms ist der Fehler noch größer.
Für genaue Messungen ist aber ohnehin die 100 ms Messzeit die bessere Wahl. Die habe ich genau ausgelegt, so dass der Fehler hier unter 0,0015% bleibt. Dieser Fehler ist kleiner als der Quarzfehler.

Messzeit und Vorteiler
Die Auflösung des Frequenzmessers ist das Produkt des Vorteilerverhältnisses mit dem Reziprok der Messzeit. Eine hohe Genauigkeit ergibt sich durch die Verwendung einer langen Messzeit (100ms) und eines möglichst kleinen Vorteilerverhältnisses.

Messzeit	**Reziprok**	**1:1**	**2:1**	**4:1**	**8:1**	**16:1**	**32:1**
1 ms	**1000 Hz**	1kHz	2kHz	4kHz	8kHz	16kHz	32kHz
10 ms	**100 Hz**	100Hz	200Hz	400Hz	800Hz	1,6kHz	3,2kHz
100ms	**10 Hz**	10Hz	20Hz	40Hz	80Hz	160Hz	320Hz

Der sich durch diese Auflösungsbeschränkung ergebende Messfehler, ist von der Frequenz abhängig. Typische Werte findet man in folgender Tabelle. Mit der 100ms-Messzeit lässt sich der Fehler unter 0,003% halten.

Messzeit	**1:1**	**4:1**	**16:1**	**32:1**
1 ms	<0,016%	<0,033%	<0,064%	<0,128%
10 ms	<0,003%	<0,003%	<0,006%	<0,128%
100ms	<0,003%	<0,003%	<0,003%	<0,003%

Erreichte Genauigkeit

	Standard-Quarz		**Quarz kalibriert**
Frequenzbereich	**garantiert**	**typisch**	**typisch**
500 kHz .. 50 MHz	0.013%	0.005%	0.003%
150 kHz .. 500 kHz	0.013%	0.006%	0.006%
50 Hz .. 150 kHz	20 Hz	10 Hz	10Hz

Maximale Eingangsfrequenz
Die Funktion des Zählfrequenzmessers ist nicht mehr garantiert, wenn die Frequenz am Vorteiler 50 MHz oder die Frequenz am 16-Bit-Zählereingang 6 MHz überschreitet. Eine Messung ist unmöglich, wenn der Zähler während der Messzeit überläuft (weil er mehr als 65535 Impulse zählen soll). Daraus ergeben sich folgende maximale Eingangsfrequenzen für verschiedene Vorteilereinstellungen:

Messzeit	**1:1**	**2:1**	**4:1**	**8:1**	**16:1**	**32:1**	**62:1**
1 ms	6 MHz	12MHz	24MHz	48MHz	50MHz	50MHz	50MHz
10 ms	6 MHz	13MHz	26 MHz	50MHz	50MHz	50MHz	50MHz
100ms	655 kHz	1,3 MHz	2,6 MHz	5,3 MHz	10,4MHz	20,9MHz	41,9MHz

7.4 RS-232

Der USB4all hat einen RS232-Port. Er erlaubt die serielle asynchrone Kommunikation mit Geschwindigkeiten von 9600 Baud bis zu 115200 Baud. Übertragen werden 8 Bit ohne Paritätsbit.

Die zwei RS232-Pins belegen Pins des PortC:
- TX = RC6
- RX = RC7

Die Pins müssen mit einem der üblichen RS232-Treiber/Pegelwandler (z.B. MAX232) versehen werden.

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um das RS232-Interface zu nutzen, muss dieses mit dem Befehl 0x01 aktiviert werden. Dabei werden die beiden Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem RS232-Interface zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des RS232-Interfaces (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

Es gibt 7 unterschiedliche Befehle für das RS232-Interface.

Byte 0 Subsystem	**Byte 1 Befehl**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**
0x59 RS232	0x00 aus	-	-	-	-
	0x01 initialisieren	0: 19200 Baud 1: 115200 Baud 2: 57600 Baud 3: 19200 Baud 4: 9600 Baud	-	-	-
	0x02 Senden eines Zeichens	Zeichen	-	-	-
	0x03 Lesen eines Zeichens	-	-	-	-
	0x04 String senden	Länge des String	1. Zeichen	2. Zeichen	.....
	0x05 String empfangen	Länge des String	-	-	-
	0x06 Anzahl der empfangenen Zeichen	-	-	-	-
	0x07 Empfangspuffer löschen	-	-	-	-

Der Befehl 0x03 wartet nicht darauf, dass ein Zeichen empfangen wird, sondern prüft den Empfangspuffer des USB4all. Befindet sich dort ein Zeichen, dann wird es gelesen. Die Quittung enthält das gelesene Zeichen im Byte 2, während das Byte 0 den Wert 0 (alles OK) hat. Befindet sich aber kein Zeichen im Puffer, dann wird trotzdem sofort eine Quittung gesendet. Im Byte 0 der Quittung wird dann als Fehlerkennzeichen 0xFF übermittelt.

USB4all antwortet auf alle Befehle mit 16-Byte. Nur bei den Befehlen 3, 5 und 6 enthalten diese Bytes eine nutzbare Information:

Byte 0	Byte 1 Befehl	Byte 2	Byte 3	Byte 4	Byte 5
0x00 oder 0xFF	0x03	Empfangenes Zeichen	-	-	-
0x59	0x05	Länge des gelesenen String	1. Zeichen	2. Zeichen	.....
0x59	0x06	Länge des String im Puffer	-	-	-

Beim Befehl 0x03 enthält das Byte 0 des empfangenen Strings eine Information über den Erfolg des RS232-Empfangs:

- 0x00 - alles in Ordnung
- 0xFF – es wurde kein Zeichen empfangen

Die Länge eines zu sendenden Strings ist beim USB4all-MCD auf 61 Zeichen und beim USB4all-CDC auf 20 Zeichen begrenzt.
Die Länge eines zu empfangenen Strings ist auf 13 Zeichen begrenzt.

USB4all hat einen 21 (CDC) bzw. 32 (MCD) Zeichen langen internen Empfangspuffer für RS232-Daten. Nach der Initialisierung des RS232-Interfaces (Befehl 1) werden alle via RS232 empfangenen Zeichen in diesen Puffer eingetragen. Die Befehle 3 und 5 lesen die Zeichen aus diesem Puffer aus.
Ist der Puffer voll, dann werden weitere eintreffende RS232-Zeichen vom USB4all solange ignoriert, bis durch einen der Befehle 3 , 5 oder 7 der Puffer wenigstens teilweise wieder geleert wurde.

Der Befehl 3 entnimmt immer nur ein Zeichen aus dem Puffer.

Der Befehl 5 versucht die gewünschte Anzahl von Zeichen auszulesen. Sind weniger Zeichen im Puffer als gewünscht, dann werden die vorhandenen Zeichen ausgelesen. Im Byte 2 wird die Anzahl der tatsächlich gelesenen Zeichen zum PC übermittelt.

Der Befehl 6 liefert die Zahl der gegenwärtig im Puffer liegenden Zeichen.

Der Befehl 7 löscht den gesamten Puffer.

7.5 I2C-Interface

Der USB4all hat ein I2C-Port. Es erlaubt die serielle Kommunikation mit anderen Schaltkreisen. Das USB4all kann nur als Master, nicht aber als Slave verwendet werden. Das I2C-Interface kann nicht gleichzeitig mit dem SPI-Interface, Schieberegister-Interface oder Microwire-Interface benutzt werden.

Die zwei I2C-Pins belegen Pins des PortB:

- SDA = RB0
- SDC = RB1

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um das I2C-Interface zu nutzen, muss dieses mit dem Befehl 0x01 aktiviert werden. Dabei werden die beiden Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem I2C-Interface zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des I2C-Interfaces (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

Es gibt 6 unterschiedliche Befehle für das I2C-Interface.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
0x54 I2C	0x00 aus					
	0x01 initialisieren	0: Master	0: 100 kHz 1: 400 kHz 2: 1 MHz			
	0x02 Senden eines Zeichens	Adresse (7 Bit)	Datenbyte			
	0x03 Lesen eines Zeichens	Adresse (7 Bit)				
	0x04 einen String schreiben	Adresse (7 Bit)	Anzahl der Datenbytes	1. Datenbyte	...	...
	0x05 einen String lesen	Adresse (7 Bit)	Anzahl der Datenbytes	-	-	-
	0x12 Senden eines Zeichens	AdresseH (obere 2 Bit)	AdresseL (untere 8 Bit)	Datenbyte		
	0x13 Lesen eines Zeichens	AdresseH (obere 2 Bit)	AdresseL (untere 8 Bit)			
	0x14 einen String schreiben	AdresseH (obere 2 Bit)	AdresseL (untere 8 Bit)	Anzahl der Datenbytes	1. Datenbyte	...
	0x15 einen String lesen	AdresseH (obere 2 Bit)	AdresseL (untere 8 Bit)	Anzahl der Datenbytes		

Die Länge eines zu schreibenden String ist auf 60 Zeichen begrenzt.
Die Länge eines zu lesenden String ist auf 12 Zeichen begrenzt.

USB4all antwortet auf alle Befehle mit 16 Byte. Nur bei den Befehlen 0x03, 0x05, 0x13 und 0x15 enthalten diese Bytes eine Information, die aus den I2C-Slaves ausgelesen wurden.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
0x00	0x03	Empfangenes Zeichen				
	0x05	Adresse	Anzahl der Datenbytes	1. Datenbyte	...	....
	0x13	Empfangenes Zeichen				
	0x15	AdresseH	AdresseL	Anzahl der Datenbytes	1. Datenbyte	....

Bei den Befehlen 0x02 ... 0x15 enthält das Byte 0 des empfangenen Strings eine Information über den Erfolg des I2C-Transfers:

- 0x00 - alles in Ordnung
- 0xFF - Bus-Kollision beim Lesen
- 0xFE - „bus device responded with NOT ACK“ beim Schreiben
- 0xFD - „return with write collision error“ beim Schreiben

Hinweis zu den 7-Bit-Adressen

Die Befehle 0x02 bis 0x05 sind für die Arbeit mit 7-Bit-Adressen vorgesehen.
Die 7-Bit-Adressen sind wirklich nur 7 Bit lang. USB4all ergänzt die Adresse dann selbständig mit einer nachgestellten 0 oder 1, um dem Slave einen Schreib- oder Lesebefehl zu übermitteln.
Der Nutzer übergibt aber USB4all im Byte 2 nur die 7-Bit-Adresse ohne nachgestelltes 8. Bit. Das MSB von Byte 2 ist Null. (“0xxxxxxx”)

Hinweis zu den 10-Bit-Adressen

Die Befehle 0x12 bis 0x15 sind für die Arbeit mit 10-Bit-Adressen vorgesehen. Da die Adresse nicht in ein Byte passt, wird sie in zwei Teile aufgeteilt. Das Byte 2 des Befehls enthält die oberen 2 Bit der 10-Bit-Adresse. Die 6 höchstwertigen Bits des Byte 2 haben den Wert 0. (“000000xx”)
Die unteren 8 Bit der Adresse werden im Byte 3 übertragen.

7.6 SPI-Interface

Der USB4all hat einen SPI-Port. Er erlaubt die serielle Kommunikation mit anderen Schaltkreisen. Das SPI-Interface kann nicht gleichzeitig mit dem I2C-Interface, Microwire-Interface, Schieberegister-Interface oder dem RS232-Interface benutzt werden.

Die SPI-Pins belegen folgende Pins des USB4all:

- SDO = RC7
- SDI = RB0
- SCK = RB1
- SS = RA5 (nur im Slave-Mode mit Slave-Select)
- CS = RB2 .. RB7 (optional im Master-Mode)

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um das SPI-Interface zu nutzen, muss dieses mit dem Befehl 0x01 aktiviert werden. Dabei werden die Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem SPI-Interface zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des SPI-Interfaces (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

Es gibt 4 unterschiedliche Befehle für das SPI-Interface.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	Byte 5
0x53 SPI	0x00 aus	-	-	-	-
	0x01 initialisieren	Anzahl der Slaves	Takt	Mode	Sample
	0x02 Senden eines Bytes	Chip select	Datenbyte	-	-
	0x03 Lesen eines Bytes	Chip select	-	-	-
	0x04 Senden mehrerer Bytes	Chip select	Anzahl der folgenden Bytes	1. Datenbyte	2. Datenbyte

USB4all antwortet auf alle Befehle mit 16-Byte. Nur beim Befehl 2 und 3 enthalten diese Bytes eine Information:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
0x53	0x02	Chip select	0 - ok 255- Fehler	-
0x53	0x03	Chip select	Empfangenes Datenbyte	-

Ausschalten (0x00)

Dieser Befehl schaltet das Interface ab. Die Pins RB1, RC7 sowie die Chip-Select-Pins werden NICHT wieder auf Input geschaltet.

Initialisieren (0x01)

Dieser Befehl initialisiert das Interface. Er stellt die Taktfrequenz sowie die Taktpolarität

und den Taktruhepegel ein. Die nötigen Daten sind in den Bytes 2 bis 5 enthalten.

Als SPI-Master kann das Interface bis zu 6 Chip-Select-Leitungen (CS) ansteuern, um 6 unterschiedliche Slaves mit dem selben SPI-Bus zu steuern. Die Zahl der gewünschten CS-Leitungen wird im Byte 2 angegeben. Dabei werden die Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem SPI-Interface zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des SPI-Interfaces (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

Byte 2	CS1	CS2	CS3	CS4	CS5	CS6
0	-	-	-	-	-	-
1	RB2	-	-	-	-	-
2	RB2	RB3	-	-	-	-
3	RB2	RB3	RB4	-	-	-
4	RB2	RB3	RB4	RB5	-	-
5	RB2	RB3	RB4	RB5	RB6	-
6	RB2	RB3	RB4	RB5	RB6	RB7

Das Interface kennt 4 Busmastermode-Taktfrequenzen sowie 2 Slave-Modes. Der gewünschte Mode wird mit dem Byte 3 eingestellt.

Byte3	Mode	Taktfrequenz
0	Default (Master)	Default (750 kHz)
1	Master	12 MHz
2	Master	3 MHz
3	Master	750 kHz
4	Reserviert - nicht benutzen!	Reserviert - nicht benutzen!
5	Slave mit Slave-Select (Pin 7)	Vom Master
6	Slave ohne Slave-Select	Vom Master

Das Interface kennt 4 Sendemodes, die sich durch Taktpolarität und Taktruhepegel unterscheiden:

Byte 4	Sendeflanke	Takt-Ruhepegel
0	Default (steigend)	Default (Low)
1	Steigend	Low
2	Fallend	High
3	Fallend	Low
4	Steigend	High

Das Interface kennt zwei unterschiedliche Zeitpunkte, an denen das SDI-Pin den Eingangspegel liest:

Byte 5	Lesezeitpunkt
0	Default (Am Ende der data-out-Zeit)
1	Am Ende der data-out-Zeit
2	In der Mitte der Data-out Zeit

Senden eines Bytes (0x02)

Mit diesem Befehl wird das Byte 3 mit dem SPI-Interface gesendet. Wurde im Byte 2 ein Slave ausgewählt, dann wird seine Chip-Select-Leitung dafür aktiviert. Ist im Byte 2 das Bit 6 gesetzt, dann wird das Chip-Select-Signal am Ende des Sendens nicht abgeschaltet.

Byte 2	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit0
Bedeutung	-	0: CS am Ende abschalten 1: CS nicht abschalten	-	-	Nummer des Slave (0 .. 6)			

Gab es beim Sendeversuch eine Buskollision, dann wird als Byte 3 zum PC der Fehlercode 255 zurückgesendet. Trat kein Fehler auf, dann wird als Byte 3 eine Null zurückgegeben.

Lesen eines Bytes (0x03)
Mit diesem Befehl wird ein Byte mit dem SPI-Interface empfangen. Wurde im Byte 2 ein Slave ausgewählt, dann wird seine Chip-Select-Leitung dafür aktiviert. Ist im Byte 2 das Bit 6 gesetzt, dann wird das Chip-Select-Signal am Ende des Sendens nicht abgeschaltet.

Byte 2	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit0
Bedeutung	-	0: CS am Ende abschalten 1: CS nicht abschalten	-	-	Nummer des Slave (0 .. 6)			

Senden mehrerer Bytes (0x04)
Mit diesem Befehl können mehrere Datenbytes gesendet werden. Die Anzahl der zu sendenen Datenbytes steht im Byte 3. Ab dem Byte 4 folgen die Datenbytes. Da ein Befehl an den USB4all maximal 64 Byte lang sein kann, können mit diesem Kommando auf einmal bis zu 60 Bytes (USB4all-MCD) bzw. 16 Bytes (USB4all-CDC) via SPI gesendet werden.

Wurde im Byte 2 ein Slave ausgewählt, dann wird seine Chip-Select-Leitung dafür aktiviert. Ist im Byte 2 das Bit 6 gesetzt, dann wird das Chip-Select-Signal am Ende des Sendens nicht abgeschaltet.
Ist im Byte 2 das Bit 7 gesetzt, dann wird das Chip-Select-Signal zwischen den einzelnen Datenbyte immer wieder kurz deaktiviert.

Byte 2	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit0
Bedeutung	0: CS blockweise 1: CS byteweise	0: CS am Ende abschalten 1: CS nicht abschalten	-	-	Nummer des Slave (0 .. 6)			

Sollen an einen Slave mehr als 60 Datenbyes (USB4all-MCD) bzw. 16 Datenbyes (USB4all-CDC) gesendet werden, dann kann das in mehreren aufeinanderfolgenden SPI-Transfers geschehen. Dabei ist jeweils im Byte 2 das Bit 6 auf 1 zu setzen, wodurch das CS-Signal zwischen den Transfers nicht abgeschaltet wird. Beim letzten Transfer ist dieses Bit nicht zu setzen, wodurch dann am Ende des letzten Transfers das CS-Signal abgeschaltet wird.

7.7 Microwire (noch nicht getestet)

Der USB4all hat einen Microwire-Port. Er erlaubt die serielle Kommunikation mit anderen Schaltkreisen. Das Microwire-Interface kann nicht gleichzeitig mit dem I2C-Interface, SPI-Interface, Schieberegister-Interface oder dem RS232-Interface benutzt werden.

Die Microwire-Pins belegen Pins des USB4all:

- SDO = RC7
- SDI = RB0
- SCK = RB1
- SS = RA5

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um das Microwire-Interface zu nutzen, muss dieses mit dem Befehl 0x01 aktiviert werden. Dabei werden die Microwire-Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem Microwire-Interface zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des Microwire-Interfaces (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

Es gibt 4 unterschiedliche Befehle für das Microwire-Interface.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4
0x67 Microwire	0x00 aus	-	-	-
	0x01 initialisieren	0: 750 kHz 1: 12 MHz 2: 3 MHz 3: 750 kHz	-	-
	0x02 Senden eines Zeichens	Zeichen	0-nicht warten 1- warten bis senden fertig	-
	0x03 Lesen eines Zeichens	low	high	-

USB4all antwortet auf alle Befehle mit 16-Byte. Nur beim Befehl 3 enthalten diese Bytes eine Information:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
0x67	0x03	Empfangenes Zeichen	-	-

7.8 Schieberegister-Interface

Der USB4all hat einen Schieberegister-Interface. Er erlaubt die serielle Kommunikation mit anderen Schaltkreisen. Das Schieberegister -Interface kann nicht gleichzeitig mit dem I2C-Interface, SPI-Interface, Microwire-Interface oder dem RS232-Interface benutzt werden.

Es handelt sich um eine Art langsames, softwaregesteuertes SPI, das es erlaubt mehrere Bytes gleichzeitig zu senden und zu empfangen. Die Datenausgangsleitung SDO wird mit dem Eingang eines Schieberegisters (z.B. 74166) oder einer Kette von Schieberegistern verbunden. Die Dateneingangsleitung SDI wird mit dem Ausgang eines Schieberegisters (z.B. 74166) oder einer Kette von Schieberegistern verbunden. Die Taktleitung SCK wird mit den Takteingängen aller Schieberegister verbunden.
Nun kann USB4all ein oder mehrere Bytes vom PC in die Schieberegisterkette hineinschieben. Gleichzeitig wird der alte Inhalt der Schieberegister in den USB4all hineingelesen und abschließend an den PC übergeben.

Die 3 Schieberegister -Pins belegen Pins des PortB:

- SDO = RC7
- SDI = RB0
- SCK = RB1

In der Praxis ist manchmal ein viertes Pin erforderlich, das ein Load-Signal an das Schieberegister abgibt. Dieses ist dann mit normalen digitalen I/O-Pins zu realisieren.

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um das Schieberegister-Interface zu nutzen, muss dieses mit dem Befehl 0x01 aktiviert werden. Dabei werden die Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem Schieberegister-Interface zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des Schieberegister-Interfaces (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

Es gibt 3 unterschiedliche Befehle für das Schieberegister -Interface (Befehl 0x02 und 0x03 sind identisch).

Subsystem	Befehl	Byte 2	Byte 3	Byte 4
0x66 Schieberegister	0x00 aus	-	-	-
	0x01 initialisieren	Mode und Takt 0x0? : 50 kHz 0x4? : 5 kHz 0x8?: 500 Hz 0xC?: 50 Hz	-	-
	0x02 oder 0x03 Senden & Empfangen von Bytes	Anzahl der Bytes	erstes Byte	zweites Byte

USB4all antwortet auf alle Befehle mit 16-Byte. Nur beim Befehl 2 und 3 enthalten diese Bytes eine Information:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
0x66	0x02 oder 0x03	Anzahl der Bytes	erstes Byte	zweites Byte

Ausschalten (0x00)
Dieser Befehl schaltet das Interface ab. Die Pins RB1 und RC7 werden NICHT wieder auf Input geschaltet.

Initialisieren (0x01)
Dieser Befehl initialisiert das Interface. Er stellt die Taktfrequenz sowie die Taktpolarität und den Taktruhepegel ein. Die nötigen Daten sind im Byte 2 enthalten.

Bit:	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Bedeutung:	FRQ1	FRQ0	-	SAMPLE	-	-	MODE1	MODE0

Das Interface kennt 4 Taktfrequenzen, die mit den Bits 6 und 7 des Bytes 2 eingestellt werden.

FRQ1	FRQ0	Taktfrequenz
0	0	50 kHz
0	1	5 kHz
1	0	500 Hz
1	1	50 Hz

Das Interface kennt 4 Sendemodes, die sich durch Taktpolarität und Taktruhepegel unterscheiden:

MODE1	MODE0	Sendeflanke	Takt-Ruhepegel
0	0	Steigend	Low
0	1	Fallend	High
1	0	Fallend	Low
1	1	Steigend	High

Das Interface kennt zwei unterschiedliche Zeitpunkte, an denen das SDI-Pin den Eingangspegel liest:

SAMPLE	Lesezeitpunkt
0	Am Ende der data-out-Zeit
1	In der Mitte der Data-out Zeit

Pinbelegung und Betriebsart wurde weitestgehend dem SPI-Interface angeglichen.

Senden und Lesen (0x02 und 0x03)
Mit diesen Befehlen (beide sind identisch) können ein oder mehrere Bytes in das Schieberegister hineingeschoben und diese gleichzeitig ausgelesen werden. Die Anzahl der Bytes steht im Byte 2. ab dem Byte 3 folgen die Datenbytes. USB4all schickt die Daten zum PC zurück, wobei die zu sendenden Daten durch die empfangenen Daten ersetzt werden.
Da immer 16 Byte zum PC gesendet werden, ist die Zahl der aus den Schieberegistern auslesbaren Bytes auf 13 begrenzt. Längere Schieberegisterketten müssen in mehreren Schritten ausgelesen werden.
Da maximal 64 Byte vom PC zum USB4all gesendet werden, ist die Zahl der schreibbaren Bytes ebenfalls begrenzt. Beim USB4all-MCD sind es 61 Bytes und beim USB4all-CDC etwa 17 Bytes.

7.9 LCD-Interface

Der USB4all kann zwei LCD-Dotmatrix-Displays mit dem HD44780-Controller ansteuern. Displays mit bis zu 2 Zeilen a 40 Zeichen bzw. 4 Zeilen a 20 Zeichen werden unterstützt. Die Interfaces werden LCD1 und LCD2 genannt. Werden beide kombiniert, lässt sich ein LCD-Display mit 4 Zeilen zu je 40 Zeichen ansteuern

Der LCD1-Anschluss belegt folgende Pins des PortB:

- E = **RB0**
- RS = RB2
- RW = RB3
- D4 = RB4
- D5 = RB5
- D6 = RB6
- D7 = RB7

Der LCD2-Anschluss belegt die selben Pins, mit Ausnahme von RB0. Der Enable-Anschluss (E) des LCD2 ist RC0:

- E = **RC0**
- RS = RB2
- RW = RB3
- D4 = RB4
- D5 = RB5
- D6 = RB6
- D7 = RB7

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um das LCD-Interface zu nutzen, muss dieses mit dem Befehl 0x01 aktiviert werden. Dabei werden die 7 Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem LCD-Interface zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des LCD-Interfaces (Befehl 0x00) sind diese Pins wieder digitale IO-Pins.

++ACHTUNG++
Das LCD1-Display kann nicht benutzt werden, wenn der I2C- oder der Microwire-Anschluss verwendet werden. Der LCD2 harmoniert dagegen mit I2C bzw. Microwire.

++ACHTUNG++
Wurden beide LCD-Interfaces initialisiert, und dann ein LCD-Interface mit dem Befehl 0x00 abgeschaltet, dann werden RB2..RB7 der Kontrolle durch das andere LCD-Interface entzogen. Das andere LCD-Display muss deshalb wieder initialisiert werden, oder man setzt die IO-Pins RB2..RB7 mit IO-Port-Befehlen auf output.

Es gibt 7 unterschiedliche Befehle für das LCD-Display.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	Byte 5
0x55 / 0x56 LCD1 / LCD2	0x00 aus	-	-	-	-
	0x01 initialisieren	Zeilenzahl	Zeichen pro Zeile	-	-

	0x02 Schreiben eines Zeichens	Zeichen	-	-	-
	0x03 Steuerbefehl	Befehls (1= Löschen)	-	-	-
	0x04 String schreiben	Länge des String	1. Zeichen	2. Zeichen	.....
	0x05 mehrere Befehle	Zahl der Befehle	1. Befehl	2. Befehl	....
	0x06 Gehe zur Position..	Zeile: 0..3	Spalte: 0..39	-	-

USB4all antwortet auf alle Befehle mit 16-Byte.

Die Länge eines zu schreibenden String ist beim USB4all-MCD auf 61 Zeichen und beim USB4all-CDC auf 20 Zeichen begrenzt.

Ausschalten (0x00)
Dieser Befehl schaltet das LCD-Interface des USB4all ab, nicht jedoch das Display selbst.

Initialisieren (0x01)
Dieser Befehl schaltet das LCD-Interface des USB4all ein, und initialisiert das daran angeschlossenen Display. Dazu müssen USB4all die Zahl der Displayzeilen (Byte 2) sowie die Zahl der Zeichen pro Zeile (Byte 3) mitgeteilt werden.
Das Anzeige des Displays wird gelöscht und der Kursor auf den Beginn der obersten Zeile gesetzt. Der Kursor selbst ist im Display nicht sichtbar.

Ein Zeichen schreiben (0x02)
Dieser Befehl schreibt das Zeichen aus dem Byte 2 auf die aktuelle Kursorposition und rückt den Kursor auf die nächste Stelle vor.

Steuerbefehl (0x03)
Es gelten die typischen Steuerbefehle des HD44780-Controllers, wie z.B.:

- Befehl 0x01 – Display löschen, Kursor auf Zeile 0/Spalte 0
- Befehl 0x02 – Kursor auf Zeile 0/Spalte 0 ohne das Display zu löschen

Mehrere Zeichen schreiben (0x04)
Dieser Befehl schreibt mehrere Zeichen in das Display. Die Anzahl der Zeichen steht im Byte 2. Ab dem Byte 3 folgen dann die zu schreibenden Zeichen.

Mehrere Steuerbefehle (0x05)
Dieser Befehl schreibt mehrere Steuerbefehle in den HD44780-Controller. Die Anzahl der Befehle steht im Byte 2. Ab dem Byte 3 folgen dann die zu schreibenden Befehle.

Gehe zur Position (0x06)
Der Befehl 0x06 kann verwendet werden, um den Kursor zum Schreiben auf eine bestimmte Stelle im Display zu setzen. Dafür wird dem USB4all die Zeilennummer und die Spaltennummer übermittelt, ab der im Folgenden geschrieben werden soll. Die Nummerierung von Zeilen und Spalten beginnt jeweils mit “0”.

Der Befehl 0x06 bezieht sich immer auf den Aufbau des Displays, das zuletzt

eingeschaltet wurde. Werden gleichzeitig zwei Displays unterschiedlichen Aufbaus am USB4all betrieben, dann kann der Befehl 0x06 also nur für das zuletzt aktivierte Display verwendet werden. Sind beide Displays identisch aufgebaut, dann funktioniert der Befehl 0x06 für beide Displays.

Die allermeisten 1-zeiligen Displays mit 16 Zeichen sind intern als Displays mit 2 Zeilen zu je 8 Zeichen organisiert. Der USB4all behandelt sie als solche. Beim Betrieb eines (sehr seltenen) echten 1x16 Displays am USB4all kann es zu vermindertem Kontrast kommen, und der Befehl 0x06 kann dann nur verwendet werden, um den Kursor auf eine der Stellen 0..7 der einzigen Zeile zu setzen.

7.10 PWM1 und PWM2

Der USB4all hat 2 PWM-Ausgänge. Mit Hilfe eines nachgeschalteten RC-Gliedes können Spannungen zwischen 0V und 5V erzeugt werden.

Die 2 PWM-Ausgänge belegen Pins des Ports C:

- PWM1 = RC2
- PWM2 = RC1

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um einen PWM-Ausgang zu nutzen, muss dieser mit dem Befehl 0x01 aktiviert werden. Dabei wird sein Pin dem digitalen IO-Pins „weggenommen“. Solange dieses Pins dem PWM zugewiesen ist, steht es nicht als digitales IO-Pins zur Verfügung. Nach dem Abschalten des PWM (Befehl 0x00) ist dieses Pins wieder ein digitales IO-Pin.

Die beiden PWM-Kanäle sind nicht unabhängig. Frequenz bzw. Periode ist bei beiden identisch. Nur das Tastverhältnis lässt sich für beide Kanäle separat einstellen. Sollen beide Kanäle gleichzeitig verwendet werden, so müssen beide identisch initialisiert werden.

Es gibt 3 unterschiedliche Befehle für jeden PWM-Kanal.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4
0x57 / 0x58 PWM1 / PWM2	0x00 aus	-	-	-
	0x01 initialisieren	0: 12 kHz / 1024 1: 480 kHz / 100 2: 120 kHz / 100 3: 30 kHz / 100 4: 187 kHz / 256 5: 47 kHz / 256 6: 12 kHz / 256 7: 47 kHz / 1024 8: 12 kHz / 1024 9: 3 kHz / 1024	-	-
	0x02 festlegen des Tastverhältnisses	Lower 8 Bits	Upper 2 Bit-	-

USB4all antwortet auf alle Befehle mit 16 Byte.

Der PWM-Ausgang kann in einer von 9 verschiedenen Moden aktiviert werden. Jeweils drei Moden haben eine Auflösung von 100 Stufen, von 256 Stufen (8 Bit) und 1024 Stufen (10 Bit). Die jeweils drei Moden gleicher Auflösung haben verschiedene Ausgangsfrequenzen.

Damit sollte für jeden was dabei sein. Wer in Prozent arbeitet, der nimmt einen 100-Stufen-Mode. Wer mit 8 Bit arbeitet wählt einen 256-Stufen-Mode. Wer es ganz präzise braucht, der nimmt einen 1024-Stufen Mode.

Mit dem Befehl 0x02 wird angegeben, für wie viele Stufen einer Periode der PWM-Ausgang high führen soll. Im 100-Stufen Mode können hier also Werte von 0 (immer low) bis 100 (immer high) eingegeben werden. Ein Tastverhältnis von 50% entspräche hier

einem Wert von 50.

Da für den 1024-Stufen-Mode Werte von bis zu 1024 übermittelt werden müssen, werden dafür im Befehl zwei Bytes benötigt. Das Byte 2 enthält die niederwertigen 8 Bit, während im Byte 3 die beiden höherwertigen Bits stehen. Im 100- bzw. 256-Stufen-Mode ist als Byte 3 immer 0x00 zu senden.

++ACHTUNG++
Falls ein anderer Bootloader als der Bootloader-5 verwendet wird, kann es zur Fehlfunktion des 2. PWM-Kanals kommen.

7.11 interner EEPROM

Der USB4all hat einen 256 Byte großen internen EEPROM, in dem Werte dauerhaft gespeichert werden können.

Nicht alle 256 Byte stehen dem Anwender zur Verfügung. Der Bootloader belegt die Zellen 0xFE und 0xFF. Den Bereich von 0xC0 bis 0xFD werde ich in späteren Firmwareversionen benötigen.
Für den Anwender verbleibt der **192 Byte große Bereich von 0x00 bis 0xBF**. Der Nutzer darf den EEPROM-Bereich von 0xC0 bis 0xFF nicht beschreiben!

Es gibt 4 unterschiedliche Befehle für den EEPROM.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	Byte 5
0x5A EEPROM	0x02 ein Byte schreiben	Adresse	Datenbyte	-	-
	0x03 ein Byte lesen	Adresse	-	-	-
	0x04 einen String schreiben	Startadresse	Anzahl der Datenbytes	1. Datenbyte	....
	0x05 einen String lesen	Startadresse	Anzahl der Datenbytes	-	-

USB4all antwortet auf alle Befehle mit 16 Byte. Nur bei den Befehlen 3 und 5 enthalten diese Bytes eine Information:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
0x5A	0x03	Empfangenes Zeichen	-	-	-
	0x05	Startadresse	Anzahl der Datenbytes	1. Datenbyte	....

Die Länge eines zu schreibenden String ist beim USB4all-MCD auf 60 Zeichen und beim USB4all-CDC auf 16 Zeichen begrenzt.
Die Länge eines zu lesenden String ist auf 13 Zeichen begrenzt.

Die EEPROM-Schreibfunktion besitzt momentan keine Sicherheitsabfrage oder Zugriffsschlüssel. Theoretisch kann ein PC (z.B. während des Bootvorgangs) zufällig eine Bytefolge wie 54-02 oder 54-04 an ein angeschlossenes USB4all senden und damit EEPROM-Daten beschädigen. Das ist zwar nicht sehr warscheinlich, trotzdem sollte man mit EEPROM-Datenverlust rechnen. Zukünftige Firmwareversionen werden einen zusätzlichen EEPROM-Aktivierungs-Befehl unterstützen, der dieses Problem beseitigen wird.

7.12 Schrittmotoransteuerung

Der USB4all kennt zwei unterschiedliche Schrittmotoransteuerungen. Die normale Schrittmotoransteuerung stellt direkt die Phasen A, B, C, D zur Verfügung, die über einen Treiber dem Motor zugeführt werden können. Alternativ gibt es die L297-Ansteuerung. Sie ist in der Lage den populären Schrittmotorencontroller L297 anzusteuern, und damit seine besonderen Features (z.B. Strombegrenzung) zu nutzen.

7.12.1 Schrittmotorenansteuerung mit ABCD-Phasen

Der USB4all hat 4 Ausgänge zur Ansteuerung von Schrittmotoren. Jedes Interface hat 4 Pins, die mit einem geeigneten Treiber verstärkt werden müssen. Die Kanäle können einzeln oder gemeinsam benutzt werden. Da es keine Spulenstromregelung gibt, nimmt das Drehmoment des Motors mit der Drehgeschwindigkeit ab.

Die 1. Schrittmotor-Ausgang (Subsystem 0x5D) belegt folgende Pins des PortB:
- A1 = RB3
- B1 = RB2
- C1 = RB1
- D1 = RB0

Die 2. Schrittmotor-Ausgang (Subsystem 0x5E) belegt folgende Pins des PortB:
- A2 = RB7
- B2 = RB6
- C2 = RB5
- D2 = RB4

Die 3. Schrittmotor-Ausgang (Subsystem 0x5F) belegt folgende Pins des PortA:
- A3 = RA0
- B3 = RA1
- C3 = RA2
- D3 = RA3

Die 4. Schrittmotor-Ausgang (Subsystem 0x5C) belegt folgende Pins des PortC:
- A4 = RC0
- B4 = RC1
- C4 = RC2
- D4 = RC6

Nach dem Einschalten der Betriebsspannung sind diese Pins als digitale Eingänge konfiguriert. Um einen Schrittmotor-Ausgang zu nutzen, muss dieser mit dem Befehl 0x01 aktiviert werden. Dabei werden seine Pins den digitalen IO-Pins „weggenommen“. Solange diese Pins dem Schrittmotor zugewiesen ist, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des Schrittmotor-Interfaces (Befehl 0x00) sind die Pins wieder digitale IO-Pins. Die beiden Schrittmotor-Interfaces können unabhängig voneinander ein- und ausgeschaltet werden.

Es gibt 4 unterschiedliche Befehle für jeden ABCD-Schrittmotorkanal.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6

0x5C..0x5F STEP1..4	0x00 aus	-	-	-	-	-
	0x01 initialisieren	-	-	-	-	-
	0x02 Motor drehen	Schrittzahl (Lower 8 Bits)	Schrittzahl (Upper 7 Bits)	Bit 0: 0–rechts 1–links Bit 1: 0-Halbschritt 1-Vollschritt Bit 2: 0-asynchron 1-synchron Bit 3: 0-keep power 1-power off Bit 4: 0-Halb-/Vollschritt 1-Wave mode Bit 5: 0-constant Geschw. 1-beschl./brems. Bit 6: 0-normal Geschw. 1-10-mal schneller	Period e [ms]	-
	0x03 Restschritte auslesen	-	-	-	-	-
	0x04 Beschleuni gungs-tabelle	0 - standard	-	-	-	-
		2 - belegen	Wert 0	Wert 1	Wert 2	....
		3 - auslesen	-	-	-	-

USB4all antwortet auf alle Befehle mit 16 Byte. Beim Befehl 0x03 enthalten diese Daten: die noch verbleibende Schritte im Asynchronmodus. Beim Befehl 0x04-0x03 enthalten sie die aktuelle Beschleunigungstabelle (8 Byte lang).

Byte 0	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**
0x5C..0x5F	0x03	Restschrittzahl (Lower 8 Bits)	Restschrittzahl (Upper 7 Bits)	-	-
0x5C..0x5F	0x04	0x03	Wert 0	Wert 1	....

0x02 Motor drehen

Das Drehen des Motors wird mit dem Befehl 0x02 ausgelöst. Der Motor kann rechts- oder links herum im Halbschritt-, Vollschritt- oder Waveverfahren gedreht werden. Festgelegt wird das im Byte 4 des Befehls.

<u>Schrittzahl</u>

Die Schrittzahl kann zwischen 1 und 32000 liegen. Die Schrittzahl wird in den Bytes 2 und

3 des Befehls 0x02 übermittelt. Wird der Motor im Halbschrittverfahren gedreht, dann bezieht sich dieser Wert auf die Anzahl der Halbschritte.

Drehverfahren
Während der Drehung werden die Pegel der Pins A, B, C und D entsprechend folgendem Schema Schritt für Schritt verändert. Die Rechtsdrehung entspricht steigender Schrittzahl, und die Linksdrehung fallender Schrittzahl.

Für den **Vollschrittmodus** gilt: A = - B sowie C = - D. Wer einen Treiberschaltkreis mit internen Invertern verwendet, findet dort nur 2 Eingänge vor. In diesem Fall werden dort nur A und C angeschlossen, während B und D nicht benötigt werden.

Ist im Byte 4 das Bit 4=1, dann wird der Motor im **Wave**-Mode angesteuert. Dabei ist immer genau ein Treiberausgang aktiv. Wurde gleichzeitig der Halbschrittmode aktiviert, dann wird der Motor nur bei jedem zweiten Schritt bewegt. Das ist bei der nötigen Schrittzahl zu berücksichtigen.

Halbschrittmode

Schritt	A	B	C	D
0	0	1	0	1
1	0	0	0	1
2	1	0	0	1
3	1	0	0	0
4	1	0	1	0
5	0	0	1	0
6	0	1	1	0
7	0	1	0	0

Vollschrittmode

Schritt	A	B	C	D
0	0	1	0	1
1 (2)	1	0	0	1
2 (4)	1	0	1	0
3 (6)	0	1	1	0

Wavemode

Schritt	A	B	C	D
0 (1)	0	0	0	1
1 (3)	1	0	0	0
2 (5)	0	0	1	0
3 (7)	0	1	0	0

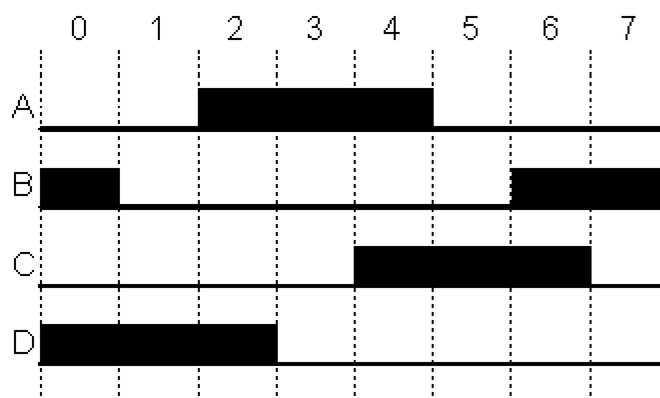


Abbildung 32 Halbschrittmode

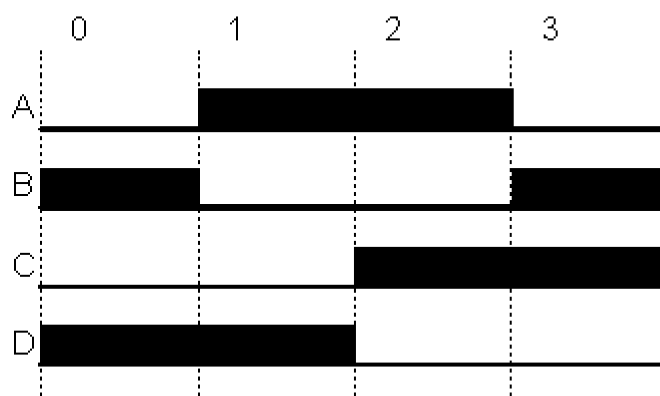


Abbildung 33 Vollschrittmode

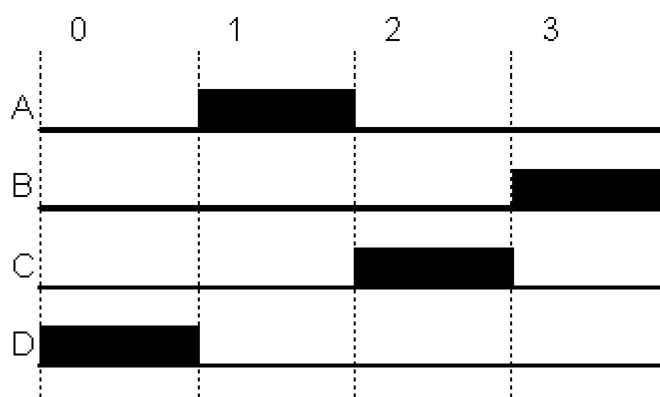


Abbildung 34 Wavemode

<u>Asynchron zum USB</u>
Ist im Byte 4 das Bit 2=0, so wird dem USB4all die Aufgabe erteilt den Motor zu drehen. Dann wird sofort die Quittung gesendet, ohne auf das Ende der Drehung zu warten, und der USB4all steht sofort für weitere Kommandos zur Verfügung, während der Motor "im Hintergrund" gedreht wird.
Mit dem Befehl 0x03 kann man jederzeit auslesen, wie viele Schritte der Schrittmotor noch machen muss, bis der Befehl endgültig abgearbeitet ist. Solange hier nicht 0 gemeldet wird, sollte man dem arbeitenden Kanal keine weiteren Aufträge erteilen, oder der laufende Auftrag würde sofort abgebrochen und durch den neuen Auftrag ersetzt.

Im asynchronen Betrieb kann man auch beide Schrittmotorenkanäle gleichzeitig betreiben. Allerdings drehen sich dann beide Motoren gleich schnell, da die Periode beider Motoren dann die Summe der beiden Einzelperioden ist.

Synchron zum USB

Alternativ gibt es den synchronen Betrieb zwischen USB-Interface und Schrittmotorsteuerung. Das bedeutet, dass erst die Drehung des Motors beendet wird, bevor die Quittung (16 Bytes) zum PC gesendet wird. (im Befehls-Byte 4 ist Bit2=1) Dabei kann es zu einem Timeout kommen, wenn man länger auf die Quittung warten muss, als die Timeout-Zeit des USB-Interfaces dauert. Wenn man als Empfangs-Timeout nur 1 Sekunde eingestellt hat, dann führen Drehungen, die länger als 1 Sekunde dauern, zu einem Timeout.
So kann man problemlos 500 Steps mit 1000 Hz abarbeiten, da das nur 0.5 Sekunden dauert. Dagegen würden 32000 Schritte mit 4 Hz (2 Stunden 13 Minuten) wohl problematisch sein.

Der synchrone Betrieb ist also die Ausnahme, und sollte nur eingesetzt werden, wenn die Motordrehung in wenigen Millisekunden abgeschlossen ist. Ansonsten verwendet man den asynchronen Betrieb, und prüft mit dem Befehl 0x03 periodisch ab, ob der Schrittzähler den Endwert 0 erreicht hat.

Power off

Ist im Byte 4 das Bit 3=1, dann werden nach dem Ende der Drehung alle Ausgänge auf “0” geschaltet. Dadurch sind die Spulen des Motors stromfrei. Zwischen dem letzten Schritt und dem Abschalten des Stroms gibt es eine kurze Wartezeit (Periode + 10 ms) damit der Motor mit Sicherheit stehen bleibt und nicht über die Sollposition hinausläuft.

Keep power

Ist im Byte 4 das Bit 3=0, dann behalten nach dem Ende der Drehung die Ausgänge A..D ihren Pegel. Dadurch fließt weiterhin Strom durch die Motorspule. Über eine kurze Zeit kann man damit den Motor in der aktuellen Stellung festhalten. Ob der Motor dadurch auf die Dauer überlastet werden würde, muss der Anwender entscheiden.

Beschleunigen und Bremsen

Ist im Byte 4 das Bit 5=1, dann wird der Motor am Anfang der Bewegung auf seine Sollgeschwindigkeit beschleunigt, und am Ende der Bewegung abgebremst. Das ermöglicht die Nutzung hoher Schrittfrequenzen ohne Schrittverlust.
Das Beschleunigen erfolgt über maximal 8 Schritte. Dabei wird der Abstand zwischen den Schritten immer weiter verringert, bis der gewünschte Schrittabstand (Periode siehe unten) erreicht ist.

Das Abbremsen erfolgt maximal über die letzten 8 Schritte der Motorbewegung. Dabei wird ausgehend von der eingestellten Periode der Schrittabstand immer weiter vegrößert.

Jeder Schrittmotorkanal hat seine eigene Tabelle mit acht Beschleunigungs/Bremswerten, die vom Anwender mit dem Befehl 0x04 ausgelesen oder verändert werden kann. Die Standardtabelle lautet:

Nr.	Wert 1	Wert 2	Wert 3	Wert 4	Wert 5	Wert 6	Wert 7	Wert 8
Schrittabstand	10 ms	8 ms	7 ms	6 ms	5 ms	4 ms	3 ms	2 ms

Beim Start des Motors werden die ersten 8 Schrittabstände aus der Tabelle entnommen, solange der vorgegebene Schrittabstand (Periode) kleiner ist als der Tabellenwert.
Am Ende des Motorlaufs wird die Tabelle während der letzten 8 Schritte rückwärts abgearbeitet.

Da die Tabellenwerte nur verwendet werden, wenn sie größer als die Soll-Periode sind, wirkt die Beschleunigung/Abbremsung nur, wenn die Periode kleiner als 10 ms ist, also bei Schrittfrequenzen über 100 Hz.
Um die Bremswerte des Kanals zu verdoppeln, sendet man in den Kanal eine neue Tabelle wie folgt: (0x5D - 0x04 - 0x02 - 20 - 16 - 14 - 13 - 10 - 8 - 6 - 4)
Um wieder den Standard herzustellen genügt (0x5D - 0x04 - 0x00).

Periode
Die Drehgeschwindigkeit wird schließlich mit dem Byte 5 (Periode) festgelegt. Der hier übertragene Wert ist der Zeitabstand zwischen zwei Schritten bzw. zwischen 2 Halbschritten in Millisekunden. Ein Wert von 1 bewirkt eine Drehtakt von 1000 Hz. Mit dem möglichen Maximalwert von 255 ergibt sich ein Drehtakt von 3,92 Hz.
Wird eine Periode=0 übertragen, dann benutzt USB4all den Standardwert 1000 Hz.

Für kleine Motoren mag 1000 Hz noch nicht das Limit darstellen. Darum gibt es noch das Bit 6 im Wort 0x04 des Befehls 0x02. Ist es gesetzt, dann wird die Schrittgeschwindigkeit verzehnfacht. Mit den möglichen Periodenwerten ergeben sich dann Schrittfrequenzen von 39 Hz bis 10000 Hz. Auch die Beschleunigungs- und Bremswerte werden dann durch 10 geteilt.

7.12.2 L297-Schrittmotorenansteuerung

Der USB4all hat 4 Ausgänge zur Ansteuerung von Schrittmotoren mit dem Schaltkreis L297. Jedes Interface hat 2 Pins, die mit dem L297 direkt verbunden werden können. Die Kanäle können einzeln oder gemeinsam benutzt werden. Die Speisung des L297, des Treiberschaltkreises und des Motors hat aus einer separaten Betriebsspannung zu erfolgen, da das USB-Interface den nötigen Strom nicht liefern kann. Die Masse des USB4all (Vss) ist mit der Masse der Motorbetriebsspannung zu verbinden.

Der 1. L297-Ausgang (Subsystem 0x60) belegt folgende Pins des PortB:
- Clock1 = RB0
- Direction1 = RB1

Der 2. L297-Ausgang (Subsystem 0x61) belegt folgende Pins des PortB:
- Clock2 = RB2
- Direction2 = RB3

Der 3. L297-Ausgang (Subsystem 0x62) belegt folgende Pins des PortB:
- Clock2 = RB4
- Direction2 = RB5

Der 4. L297-Ausgang (Subsystem 0x63) belegt folgende Pins des PortB:
- Clock2 = RB6
- Direction2 = RB7

Das Pin des L297 für Halbschritt/Vollschritt-Umschaltung ist fest mit dem gewünschten Potential zu verbinden. Das Enable-Pin ist mit High-Pegel (Vdd) zu verbinden.

Nach dem Einschalten der Betriebsspannung sind die Clock- und Direction-Pins als digitale Eingänge konfiguriert. Um einen L297-Ausgang zu nutzen, muss dieser mit dem Befehl 0x01 aktiviert werden. Dabei werden seine Pins den digitalen IO-Pins

„weggenommen“. Solange diese Pins dem Schrittmotor zugewiesen sind, stehen sie nicht als digitale IO-Pins zur Verfügung. Nach dem Abschalten des L297-Interfaces (Befehl 0x00) sind die Pins wieder digitale IO-Pins. Die vier L297-Interfaces können unabhängig voneinander ein- und ausgeschaltet werden.

Es gibt 4 unterschiedliche Befehle für jeden L297-Kanal. Sie entsprechen den Befehlen für die ABCD-Phasen-Schrittmotorsteuerung, wobei nicht notwendige Kommandos entfernt wurden.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
0x60 .. 0x63 L297_1 .. L297_4	0x00 aus	-	-	-	-	-
	0x01 initialisieren	-	-	-	-	-
	0x02 Motor drehen	Schrittzahl (Lower 8 Bits)	Schrittzahl (Upper 7 Bits)	Bit 0: 0–rechts 1–links Bit 2: 0-asynchron 1-synchron	Periode [ms]	0x00 (reserviert)
	0x03 Restschritte auslesen	-	-	-	-	-

USB4all antwortet auf alle Befehle mit 16 Byte. Nur beim Befehl 0x03 enthalten diese Daten: die noch verbleibende Schritte im Asynchronmodus.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
0x60 / 0x63	0x03	Restschrittzahl (Lower 8 Bits)	Restschrittzahl (Upper 7 Bits)	-	-

0x02 Motor drehen

Das Drehen des Motors wird mit dem Befehl 0x02 ausgelöst. Der Motor kann rechts- oder linksherum gedreht werden. Festgelegt wird das im Byte 4 des Befehls.

Die Schrittzahl kann zwischen 1 und 32000 liegen. Die Schrittzahl wird in den Bytes 2 und 3 des Befehls 0x02 übermittelt. Wird der Motor im Halbschrittverfahren gedreht, dann bezieht sich dieser Wert auf die Anzahl der Halbschritte.

Asynchron zum USB

Ist im Byte 4 das Bit 2=0, so wird dem USB4all die Aufgabe erteilt den Motor zu drehen. Dann wird sofort die Quittung gesendet, ohne auf das Ende der Drehung zu warten, und der USB4all steht sofort für weitere Kommandos. zur Verfügung.
Mit dem Befehl 0x03 kann man jederzeit auslesen, wie viele Schritte der Schrittmotor noch machen muss, bis der Befehl endgültig abgearbeitet ist. Solange hier nicht 0 gemeldet wird, sollte man dem arbeitenden Kanal keine weiteren Aufträge erteilen.

Im asynchronen Betrieb kann man auch alle Schrittmotorenkanäle gleichzeitig betreiben. Allerdings drehen sich dann alle Motoren gleich schnell, da die Periode aller Motoren dann die Summe der Einzelperioden ist.

Synchron zum USB
Alternativ gibt es den synchronen Betrieb zwischen USB-Interface und Schrittmotorsteuerung. Das bedeutet, dass erst die Drehung des Motors beendet wird, bevor die Quittung (16 Bytes) zum PC gesendet wird (im Byte 4 ist Bit 2=1). Dabei kann es zu einem Timeout kommen, wenn man länger auf die Quittung warten muss, als die Timeout-Zeit des USB-Interfaces dauert. Wenn man als Empfangs-Timeout nur 1 Sekunde eingestellt hat, dann führen Drehungen, die länger als 1 Sekunde dauern, zu einem Timeout.
So kann man problemlos 500 Steps mit 1000 Hz abarbeiten, da das nur 0.5 Sekunden dauert. Dagegen würden 32000 Schritte mit 4 Hz (2 Stunden 13 Minuten) wohl problematisch sein.

Der Synchrone Betrieb ist also die Ausnahme, und sollte nur eingesetzt werden, wenn die Motordrehung in wenigen Millisekunden abgeschlossen ist. Ansonsten verwendet man den asynchronen Betrieb, und prüft mit dem Befehl 0x03 ab, ob der Schrittzähler den Endwert 0 erreicht hat.

Periode
Die Drehgeschwindigkeit wird schließlich mit dem Byte 5 (Periode) festgelegt. Der hier übertragene Wert ist der Zeitabstand zwischen zwei Schritten bzw. zwischen 2 Halbschritten in Millisekunden. Ein Wert von 1 bewirkt eine Drehtakt von 1000 Hz. Mit dem möglichen Maximalwert von 255 ergibt sich ein Drehtakt von 3,92 Hz.
Wird eine Periode=0 übertragen, dann benutzt USB4all den Standardwert 1000 Hz.

7.13 Servos

Der USB4all hat 13 Ausgänge für Modellbauservos. Man kann entweder alle 13 Kanäle benutzen, oder sich auf einige Kanäle beschränken.

Die Servos sind in zwei unabhängige Gruppen aufgeteilt, die separat programmiert werden müssen. Acht Servos (SB0 ... SB7) gehören zur Gruppe Servo-B (am Port B) und die anderen 5 Servos (SC0, SC1, SC2, SC6, SC7) zur Gruppe Servo-C (am Port C).

Aktive Kanäle erzeugen positive Rechteckpulse, deren Pulsbreite im Bereich von 1 ms bis zu 2 ms in 100 Abstufungen eingestellt werden kann.
Die Pulswiederholrate liegt bei Nutzung der Gruppe-B bei etwa 50 Hz und bei Nutzung der Gruppe-C bei 80 Hz. Werden beide Gruppen verwendet, fällt die Pulswiederholrate auf 30 Hz.

Es gibt 4 unterschiedliche Befehle:

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	... Byte
0x64 Servo B oder 0x65 Servo C	0x00 aus	-	-	-	-
	0x01 initialisieren	Bitmaske: 0: Pin aus 1: Pin aktiv	-	-	-
	0x02 festlegen des Tastverhältnisses	Servo SB0 oder Servo SC0	Servo SB1 oder Servo SC1	Servo SB2 oder Servo SC2	Servo SB... oder Servo SC...
	0x03 Festlegen des Nullpunktes	Nullpunkt	-	-	-

USB4all antwortet auf alle Befehle mit 16-Byte.

0x00 Servos aus
Alle Kanäle der Gruppe beenden die Erzeugung von Pulsen. Pins, die bisher aktiv waren, geben nun permanent Low-Pegel aus.

0x01 initialisieren
Es wird festgelegt, welche Pins der Servo-Gruppe Servo-Pulse erzeugen sollen. Dafür wird eine Bitmaske an USB4all übertragen (Byte 2). Jedes Bit in diesem Byte entspricht einem Servo-Pin. Eine “1” im jeweiligen Bit aktiviert den zugehörigen Servo-Ausgang. Das Pin wird automatisch auf “Ausgabe” eingestellt, und beginnt mit der Erzeugung von Pulsen der Länge 1,5 ms. Sind bereits beim Beginn der Pulserzeugung andere Pulsbreiten erforderlich, so sind diese unmittelbar vorher mit dem Befehl **0x02** einzustellen.
Pins, deren zugehörige Bits in der Maske auf “0” stehen, werden von der Servo-Steuerung ignoriert, und können für andere Interfaces verwendet werden.

Jedes IO-Pin des Port B ist einem Bit im Maskenbyte von Servo-B wie folgt zugeordnet.

Maske	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Servo	SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0

Jedes IO-Pin des Port C ist einem Bit im Maskenbyte von Servo-C wie folgt zugeordnet.

Maske	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin	RC7	RC6	-	-	-	RC2	RC1	RC0
Servo	SC7	SC6	-	-	-	SC2	SC1	SC0

0x02 Servo-Stellungen festlegen
Die Servoposition (Ausschlag) wird durch die Länge der ausgegebenen Pulse gesteuert. Diese wiederum werden durch 8 Steuerbytes bestimmt. Jedem Servo-Ausgang ist ein Steuerbyte zugeordnet, das einen Wert von 0 bis 100 einnehmen darf. Der Wert 0 bewirkt den unteren Servoendanschlag und 100 den oberen Servoendanschlag. Ein Wert von 50 bewirkt die Servomittelstellung.
Steuerbytes , die einem inaktiven Kanal zugeordnet sind, werden vom USB4all ignoriert, müssen aber übertragen werden. Im Befehl für Servo-C gibt es drei funktionslose Füllbytes (Byte 5 .. 7), die auch übertragen werden müssen.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x64	0x02	SB0	SB1	SB2	SB3	SB4	SB5	SB6	SB7
0x65	0x02	SC0	SC1	SC2	-	-	-	SC6	SC7

Falls ein Wert von 100 nicht ausreicht, um den oberen Servoendanschlag des Servos zu erreichen, kann auch ein höherer Wert benutzt werden. Das Maximum liegt bei 255, wodurch eine Pulsbreite von ca 3,5 ms erreicht werden würde.

0x03 Nullpunkt festlegen
Eigentlich sollte bei einer Servostellung von 0 (Befehl 0x02) der untere Servoendanschlag erreicht werden, da dabei eine Pulsbreite von 1ms eingestellt ist. Manche Servos benötigen aber etwas kürzere Pulse (z.B. 0,8 ms). Um die Minimalpulsweite zu vermindern, muss der Nullpunkt verstellt werden. Das erfolgt mit dem Befehl **0x03**.
Das Byte 2 des Befehls enthält die minimale Pulslänge für alle Servos in 5-us-Stufen. Ein Wert von 200 (0xC8) entspricht also 200 x 5us = 1ms und ein Wert von 160 (0xA0) entspricht 0,8 ms.
Ein veränderter Nullwert wird nicht dauerhaft gespeichert, bei einem Reset oder per Power-on wird er jedesmal mit ein Standardwert von 1 ms überschrieben.

++ACHTUNG++
Die Servosteuerung unterbricht alle anderen laufenden Prozesse für die Dauer der positiven Pulse (min 13 ms bis zu max. 33 ms lang). Gleichzeitig asynchron drehende Schrittmotoren werden auf einen Schritt pro Servozyklus verlangsamt. (80 Hz .. 30 Hz)

7.14 Impulszähler

Der USB4all hat 2 Inpulszähler (Counter_0 und Counter_3), die die an zwei Pins (RA4 bzw. RC0) eintreffenden Impulse zählen können.

- Counter_0 - RA4
- Counter_3 - RC0

Counter_0 erhöht seinen Zählerstand immer bei jeder fallenden Flanke an RA4. Counter_ 3 erhöht seinen Zählerstand immer bei jeder steigenden Flanke an RC0.
Der High- und der Low-Teil eines Impulses sollten mindestens je 60 ns betragen. Beide Pins haben Schmitt-Trigger-Eingänge. Der Eingangs-Low-Pegel muss unter 1V und der High-Pegel über 4V liegen.

Beide Zähler sind 16-Bit weit, können also bis maximal 65535 zählen. Treffen danach weitere Impulse ein, dann fängt der Impulszähler wieder beim Zählerstand Null an.

Nach der Initialisierung stehen die Zähler auf Null, man kann sie aber auch auf gewünschte Werte voreinstellen.

Es gibt 5 unterschiedliche Befehle:

Subsystem	Befehl	Byte 2	Byte 3
0x68 Counter_0 oder 0x69 Counter_3	0x00 aus	-	-
	0x01 initialisieren	-	-
	0x02 Zählerstand auslesen	-	-
	0x03 Zählerstand einstellen	Low-Byte	High-Byte
	0x04 Zählerstand zurücksetzen	-	-

USB4all antwortet auf alle Befehle mit 16-Byte. Nur nach dem Befehl 0x02 enthalten diese den aktuellen Zählerstand
.

Byte 0	Byte 1	Byte 2	Byte 3
0x68 / 0x69	0x02	Zählerstand (Lower 8 Bits)	Zählerstand (Upper 8 Bits)

0x00 Zähler aus
Der Zähler wird abgeschaltet. Das zugehörige Pin steht nun wieder anderen Funktionen zur Verfügung.

0x01 Zähler initialisieren
Der Zähler wird aktiviert. Das zugehörige Pin wird auf “input” gestellt und für andere Interfaces gesperrt. Der 16-Bit Zähler wird auf den Wert Null eingestellt.

0x03 Zählerstand auslesen
Der 16-Bit Zählerstand wird ausgegeben.

0x04 Zählerstand einstellen
Der Zählerstand wird auf einen gewüschten 16-Bit-Wert eingestellt. Dieser wird im Byte 2 und Byte 3 mit übergeben. Der Zähler zählt dann ab diesem neuen Wert weiter.

0x04 Zählerstand zurückstzen
Der Zählerstand wird auf Null zurückgesetzt. Der Zähler bleibt aktiv und zählt dann ab diesem neuen Wert weiter.

7.15 Reset des USB4all

Der USB4all kann mit einem Befehl auf den Initialisierungszustand zurückgesetzt werden. Dabei meldet dich der USB4all vom USB-Bus ab und danach wieder an.

Subsystem	Befehl	Byte 2	Byte 3	Byte 4	Byte 5
0xFF RESET	-	-	-	-	-

USB4all antwortet auf diesen Befehl **NICHT** via USB.

++ACHTUNG++
Beim USB4all-CDC funktioniert die Reset-Funktion nicht korrekt. Der Aufruf der Funktion führt zum Absturz der USB4all-CDC-Firmware.

8 Ansteuerung des USB4all

Um den USB4all zu nutzen, muss man nur in der Lage sein, Strings (also kurze Bytefolgen) in den USB4all zu schreiben, und auszulesen. Beim USB4all-MCD erfolgt das mit Hilfe von Funktionen die eine DLL bereitstellt. Beim USB4all-CDC erfolgt dagegen die Kommunikation über eine emulierte, serielle RS232-Schnittstelle.

8.1 USB4all-CDC

Wem der Umgang mit DLL-Funktionen zu kompliziert ist, für den bietet sich USB4all-CDC an. Es wird so angesteuert, als wenn es sich um ein z.B. am COM3-Port angeschlossenes Gerät mit einer RS232-Schnittstelle handeln würde. (Je nach PC kann es auch mal COM4 sein. Man prüft nach dem Anstecken des USB4all an den PC am besten im Gerätemanager, unter welcher Com-Port-Nummer Windows das Device verwaltet.) Man kann es notfalls sogar mit einem einfachen Terminalprogramm (z.B. Hyper Terminal) ansteuern.

Da der Datenfluss über eine RS232-Schnittstelle zeichenweise erfolgt (und nicht wie über USB blockweise), muss es ein Verfahren geben, um den Anfang und das Ende eines Befehlsstrings zu kennzeichnen. Um das zu vereinfachen, ist es üblich nicht direkt die Datenbytes zu übertragen, sondern ASCII-Zeichen. Anstelle des hexadezimalen Bytes 0x4A wird der ASCII-String ‘4A’ übertragen, der aus zwei Zeichen besteht. Um Datenbytes voneinander zu trennen, sendet man zwischen den Wertestrings noch Trennzeichen. Das bläst den Datenstrom zur dreifachen Größe auf. Der Datentransfer läuft mit bis zu 1 Mbit/s aber ausreichend schnell.

Den Beginn eines Kommandostrings kennzeichnet ein ‘#’.

Das Ende eines Kommandostrings kennzeichnet ein Byte mit dem Wert 0x00 (nullterminiert), 0x0A (Zeilenvorschub) oder 0x0D (Wagenrücklauf). Die Gesamtlänge des String darf nicht größer als 64 sein. Da für ein Datenbyte im String drei ASCII-Zeichen verwendet werden, ist die Befehlslänge auf 20 Datenbyes begrenzt, was aber nur bei einigen Blocktranfer-Befehlen eine Einschränkung darstellen könnte.

Beispiel:
Um das LCD1 zu initialisieren muß man folgende 4 Datenbytes zum USB4all senden: 0x55, 0x01, 0x02, 0x16

Für das USB4all-CDC sendet man dafür folgenden 13 Byte langen String: ‘#55-01-02-16’+0x00

Das nachgestellte +0x00 bedeutet, dass dem String aus ASCII-Zeichen ein 0-Byte folgt. Als Trennzeichen habe ich hier ‘-‘ verwendet. Man kann aber prinzipiell alle druckbaren Zeichen verwenden mit Ausnahme von #,0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,A,B,C,D,E,F. Anstelle von ’01’ kann man auch einfach ‘1’ schreiben, führende Nullen können also entfallen.

Die Wandlung eines Byte-Strings in einen ASCII-String demonstriert folgender Codeschnipsel (Delphi/Pascal). Am Anfang stehen die Bytes in send_buf, am Ende steht der ASCII-String in asciistr, welcher mit Comport1.WriteStr an den USB4all-CDC gesendet wird.

```
  var
    k           : integer;
    asciistr    : string;
    send_buf    : array[0..63] of byte;
.
.
.
    asciistr :='#';
    for k:=0 to 15 do asciistr := asciistr + inttohex(send_buf[k],2)+'-';
    asciistr := asciistr + chr($0d);
    Comport1.WriteStr(asciistr);
```

Das USB4all-CDC antwortet ebenfalls mit (terminalprogrammtauglichen) ASCII-Strings. Das Startzeichen ist hier ein ‘@’, das Trennzeichen ein ‘-‘ und am Ende steht die 3 Byte lange Folge 0x0A,0x0D,0x00. In den hexadezimalen Zahlen werden immer Grossbuchstaben (A..F) verwendet. Da immer 16-Datenbytes im Antwort-String übertragen werden, und hier führende Nullen nicht unterdrückt werden, ist die Stringlänge insgesamt 50 Byte.

Wenn man Werte aus dem USB4all empfängt (z.B. Messwerte vom ADC), dann muss man sie wieder aus dem ASCII-String herauslesen. Das ist recht umständlich. Diesen Preis bezahlt man für die einfache Benutzung der virtuellen RS232-Schnittstelle.

Da aber der Aufbau des empfangenen Strings fest ist, wird aber auch die Rückwandlung in einen Bytestring nicht zu kompliziert, wie folgendes Codebeispiel in Pascal/Delphi zeigt::

```
//Hilfsroutine: Wandeln eines ASCII-Zeichens in einen Zahlenwert
//wäre mit ORD einfacher zu schreiben, ist aber so übersichtlicher
function asciiwert(ch:char):integer;
begin
  result:=0;
  case ch of
  '1': result:=1;
  '2': result:=2;
  '3': result:=3;
  '4': result:=4;
  '5': result:=5;
  '6': result:=6;
  '7': result:=7;
  '8': result:=8;
  '9': result:=9;
  'A': result:=10;
  'B': result:=11;
  'C': result:=12;
  'D': result:=13;
  'E': result:=14;
  'F': result:=15;
  end;
end;
.
.
.
  var
    nrrx        : integer;
    k           : integer;
    rxstr       : string;
    receive_buf : array[0..63] of byte;
.
.
.
    //empfangen über comport mit ReadStr(var Str: String; Count: Integer): Integer;
```

```
    nrrx:=Comport1.ReadStr(rxstr, 50);
    for k:=0 to (nrrx div 3)-1 do begin
      receive_buf[k]:=asciiwert(rxstr[3*k+2])*16 + asciiwert(rxstr[3*k+3]);
    end;
.
.
.
```

Die Nutzung des seriellen Ports sollte aus jeder Programmiersprache heraus möglich sein. Ich benutze unter Delphi die freie ComPort-Komponente von Dejan Crnila.

8.2 USB4all-MCD

Um den USB4all-MCD zu nutzen, muss man nur in der Lage sein, Strings (also kurze Bytefolgen) in den USB4all zu schreiben, und auszulesen. Dafür habe ich mir unter Delphi folgende kleine Routine geschrieben, die aus einem Byte-Array (send_buf) N-Bytes zum USB4all-MCD schreibt, und dann von dort wieder M-Bytes erwartet (im Byte-Array receive_buf). Die hier benutzten Funktionen werden von der **mpusbapi.dll** bereitgestellt. Eine detaillierte Beschreibung der mpusbapi.dll-Funktionen findet man im PDF-Handbuch zum USB-FS-Testboard von Microchip.
Wie man sie anwendet, kann man auch in meinem Beispielprogramm USB-Test (auf www.sprut.de) sehen.

```
//N Bytes senden und M Bytes empfangen
//timeout ist 100 ms bzw 1s
procedure Sende_Empfange(N,M :byte);
var
   selection   : DWORD;
   RecvLength  : DWORD;
   fehler      : integer;
begin
  selection:=0;

  myOutPipe:= _MPUSBOpen(selection,vid_pid,out_pipe,MP_WRITE,0);
  myInPipe := _MPUSBOpen(selection,vid_pid,out_pipe,MP_READ,0);
  if ((myOutPipe = INVALID_HANDLE_VALUE) or (myInPipe = INVALID_HANDLE_VALUE)) then
  begin
    info('USB Error, no pipes');
    exit;
  end;
  RecvLength:=M;
  fehler:=SendReceivePacket(send_buf,N,receive_buf,RecvLength,100,1000);
  if Fehler<>1 then  info('USB Error :'+inttostr(fehler));

  _MPUSBClose(myOutPipe);
  _MPUSBClose(myInPipe);
  myInPipe:= INVALID_HANDLE_VALUE;
  myOutPipe:=INVALID_HANDLE_VALUE;
end; // sende_empfange
```

Nach dem ich am Programmanfang prüfe, ob auch wirklich ein USB4all-MCD am PC angeschlossen ist, benutze ich nur noch diese eine Funktion (**`Sende_Empfange(N,M)`**) für die Kommunikation mit dem USB4all-MCD.

8.3 Beispiele für die Ansteuerung des USB4all

Im folgenden einige Beispiele zur Nutzung des USB4all. Alle Codeschnipsel stammen aus Delphi/Pascal-Programmen.

8.3.1 Beispiel: Schreiben eines Bytes in den EEPROM

Der folgende Programmschnipsel schreibt in die EEPROM-Zelle mit der Adresse 0x00 den Wert 0x55.

USB4all-MCD:

```
send_buf[0]:=$5A;       // EEPROM
send_buf[1]:=2;         // schreiben
send_buf[2]:=0;         // Adresse   = 0
send_buf[3]:=$55;       // Datenbyte = 0x55
Sende_Empfange(16, 16);
```

USB4all-CDC:

```
Comport1.WriteStr(‘#5A-2-0-55’+chr(0));       // EEPROM
Comport1.ReadStr(rxstr, 50);                  // Quittung
```

8.3.2 Beispiel: Messen einer Spannung

Der folgende Programmschnipsel initiiert den ADC mit den Eingängen AN0 .. AN3 und misst die Spannung am Pin AN2.

USB4all-MCD:

```
send_buf[0]:= $51;      // ADC
send_buf[1]:=1;         // initialisieren
send_buf[2]:=4;         // AN0..AN3
send_buf[3]:=0;
Sende_Empfange(16, 16);

send_buf[0]:= $51;      // ADC
send_buf[1]:=2;         // set AN
send_buf[2]:=2;         // AN2 ist der aktive Eingang
Sende_Empfange(16, 16);

send_buf[0]:= $51;      // ADC
send_buf[1]:=3;         // Spannung messen
Sende_Empfange(16, 16);
Spannung:=receive_buf[3];                  //high (obere  2 Bit)
Spannung:= Spannung *256+ receive_buf[2];  //low  (untere 8 Bit)
```

USB4all-CDC:

```
Comport1.WriteStr(‘#51-1-4-0’+chr(0));        // initialisieren
Comport1.ReadStr(rxstr, 50);                  // Quittung

Comport1.WriteStr(‘#51-2-2’+chr(0));          // set AN2
Comport1.ReadStr(rxstr, 50);                  // Quittung

Comport1.WriteStr(‘#51-3’+chr(0));            // Spannung messen
Comport1.ReadStr(rxstr, 50);                  // Quittung mit Messwert
```

8.3.3 Beispiel: Messen einer Frequenz

Der folgende Programmschnipsel misst die Frequenz eines Signals am Pin RA4.

USB4all-MCD:

```
send_buf[0]:= $52;      // Frequenzmesser
send_buf[1]:=5;         // messen mit Autorange
Sende_Empfange(16, 16);

Frequenz:=receive_buf[5];                 //high (obere   8 Bit)
Frequenz:= Frequenz *256+ receive_buf[4];  //next (nächste 8 Bit)
Frequenz:= Frequenz *256+ receive_buf[3];  //next (nächste 8 Bit)
Frequenz:= Frequenz *256+ receive_buf[2];  //low  (untere  8 Bit)
```

USB4all-CDC:

```
Comport1.WriteStr('#52-5'+chr(0));            // Frequenzmessung
Comport1.ReadStr(rxstr, 50);                  // Quittung mit Messwert
```

8.3.4 Beispiel: Ausgabe von “Hallo” am LCD-Display

Der folgende Programmschnipsel initiiert das LCD und gibt den Text”Hallo” aus:

USB4all-MCD:

```
send_buf[0]:=$55;          // LCD
send_buf[1]:=1;            // init
send_buf[2]:=2;            // 2  Zeilen
send_buf[3]:=16;           // 16 Zeichen pro zeile
Sende_Empfange(16, 16);

send_buf[0]:=$55    ;   // LCD
send_buf[1]:=4;         // String schreiben
send_buf[2]:=5;         // 5 Zeichen lang
send_buf[3]:=ord('H');  // 'H'
send_buf[4]:=ord('a');  // 'a'
send_buf[5]:=ord('l');  // 'l'
send_buf[6]:=ord('l');  // 'l'
send_buf[7]:=ord('o');  // 'o'
Sende_Empfange(16, 16);
```

USB4all-CDC:

```
Comport1.WriteStr('#55-1-2-10'+chr(0));       // initialisieren
Comport1.ReadStr(rxstr, 50);                  // Quittung

Comport1.WriteStr('#55-4-5-48-61-6c-6c-6f'+chr(0));          // Hallo
Comport1.ReadStr(rxstr, 50);                  // Quittung
```

8.3.5 Beispiel: Einschalten einer LED am Pin RC0

Der folgende Programmschnipsel schaltet das Pin RC0 auf “output” und gibt dort high Pegel aus:

USB4all-MCD:

```
send_buf[0]:=$50;       // IO-Pins
send_buf[1]:=5;         // Pin auf output setzen
```

```
    send_buf[2]:=0;          // TRISA: keines
    send_buf[3]:=0;          // TRISB: keines
    send_buf[4]:=1;          // TRISC: nur Pin RC0
    Sende_Empfange(16, 16);

    send_buf[0]:=$50;        // IO
    send_buf[1]:=6;          // Pin auf high setzen
    send_buf[2]:=0;          // TRISA: keines
    send_buf[3]:=0;          // TRISB: keines
    send_buf[4]:=1;          // TRISC: nur Pin RC0
    Sende_Empfange(16, 16);
```

USB4all-CDC:

```
    Comport1.WriteStr('#50-5-0-0-1'+chr(0));      // RC0 auf Ausgang schalten
    Comport1.ReadStr(rxstr, 50);                  // Quittung

    Comport1.WriteStr('#50-6-0-0-1'+chr(0));      // RC0 High-Pegel einschalten
    Comport1.ReadStr(rxstr, 50);                  // Quittung
```

8.3.6 Beispiel: Drehen eines Schrittmotors

Der folgende Programmschnipsel initiiert das 2. ABCD-Phasen-Schrittmotorinterface und dreht den angeschlossenen Motor um 10000 Halbschritte links herum.

USB4all-MCD:

```
    send_buf[0]:=$5E;          // 2 Schrittmotorkanal
    send_buf[1]:=1;            // on
    Sende_Empfange(16, 16);

    send_buf[0]:=$5E;
    send_buf[1]:=2;            // dreh
    send_buf[2]:=$10;          // 10000 low-Teil
    send_buf[3]:=$27;          //       high-Teil
    send_buf[4]:=1;            // links, halbschritte, asynchron, power-off
    send_buf[5]:=0;            // 1000 Hz
    Sende_Empfange(16, 16);

    // warten auf das Ende der Drehung
    Repeat
        Sleep(10);                       // 10ms Pause
        send_buf[0]:=$5E;
        send_buf[1]:=3;                  // lese Restschrittzahl
        Sende_Empfange(16, 16);
        wert:=receive_buf[3];            // high
        wert:=wert*256+ receive_buf[2]; // low
    until (Wert < 1);
```

USB4all-CDC:

```
    Comport1.WriteStr('#5E-1'+chr(0));            // initialisieren
    Comport1.ReadStr(rxstr, 50);                  // Quittung

    Comport1.WriteStr('#5E-2-10-27-1-0'+chr(0));  // drehen
    Comport1.ReadStr(rxstr, 50);                  // Quittung

    //es fehlt hier noch die Warteschleife, falls man warten möchte
```

8.3.7 Beispiel: Temperaturmessung mit einem LM75 via I2C

Der folgende Programmschnipsel initiiert das I2C-Interface liest die Temperatur aus einem angeschlossenen LM75-Temperatursensor aus.

USB4all-MCD:

```
send_buf[0]:=$54;
send_buf[1]:=1;           // on
send_buf[2]:=0;           // Master
send_buf[3]:=0;           // 100 kHz
Sende_Empfange(16, 16);

send_buf[0]:=$54;
send_buf[1]:=3;           // string lesen
send_buf[2]:=$48;         // Adresse des LM75 = 100_1000
send_buf[3]:=2;           // 2 Bytes lesen
Sende_Empfange(16, 16);
; receive_buf[4] enthält die Temperatur in Grad
; receive_buf[5] enthält die Nachkommastelle
```

USB4all-CDC:

```
Comport1.WriteStr('#54-1-0-0'+chr(0));        // on Master 100kHz
Comport1.ReadStr(rxstr, 50);                  // Quittung

Comport1.WriteStr('#54-3-48-2'+chr(0));       // Temperatur auslesen
Comport1.ReadStr(rxstr, 50);                  // Quittung mit Temperaturwert
```

8.3.8 Beispiel: Reset des USB4all

Der folgende Programmschnipsel setzt den USB4all-MCD in den Anfangszustand (wie nach dem Einschalten der Betriebsspannung) zurück, und meldet ihn neu am USB-Bus an.

USB4all-MCD:

```
send_buf[0]:= $FF;          //RESET DEVICE;
Sende_Empfange(1,0);
```

8.4 Nutzung des USB4all-MCD unter Linux

Natürlich kann man USB4all auch unter Linux nutzen. Dieser Abschnitt basiert wesentlich auf Beispiele, die Roland Wundrig erstellt hat.

Um auf USB4all-MCD zuzugreifen, eignet sich libusb (http://libusb.wiki.sourceforge.net/), was man am besten durch den Paketmanager der Linuxdistribution installieren lässt.

Da der Zugriff auf USB4all nicht über Kernelmodule, sondern direkt über libusb erfolgt, sind eigentlich root-Rechte erforderlich. Damit alle User Zugriff bekommen legt man eine Datei "**/etc/udev/rules.d/99-sprutbrenner.rules**" an und schreibt in diese Datei:

```
SUBSYSTEM=="usb", SYSFS{idProduct}=="ff0b", SYSFS{idVendor}=="04d8",GROUP =
"plugdev"
```

Damit sollte der USB4all-MCD ab dem nächsten Einstecken für alle User der plugdev-Gruppe (in der man ja grundsätzlich sein muss, wenn man USB-Geräte benutzen möchte) verfügbar sein. Dieser Eintrag ermöglicht auch den Zugriff auf andere von mir erstellte USB-Geräte, daher der nicht ganz passende Filename, den man natürlich ändern kann.

Die Files **usb4all.h** und **usb4all.c** stellen die für die Nutzung des USB4all-MCD nötigen Funktionen und Datenstrukturen bereit.

- `int usb4all_initialize(struct usb4all_t *usb4con)`
- `int usb4all_connect(struct usb4all_t *usb4con)`
- `int usb4all_data_io(struct usb4all_t *usb4con, unsigned char *data_in, int data_in_cnt, unsigned char *data_out, int data_out_cnt)`
- `int usb4all_finalize(struct usb4all_t *usb4con)`

usb4all_initialize
Diese Funktion initialisiert libusb und die Datenstruktur.

usb4all_connect
Diese Funktion sucht nach dem angeschlossenen USB4all.

usb4all_data_io
Diese Funktion dient nun dem eigentlichen Datenaustausch mit USB4all-MCD

usb4all_finalize
Diese Funktion schließt die Verbindung zum USB4all.

Programmbeispiele mit den Quelltexten finden sich im ZIP-File von USB4all im Unterordner"linux".

9 Bootloader

Hin und wieder wird für den USB4all eine neue Firmware veröffentlicht. Das ist nötig, um gefundene Fehler in der Firmware zu beseitigen oder um die Möglichkeiten des USB4all zu erweitern.
Der Bootloader ermöglicht es, die Firmware einfach und schnell zu aktualisieren.

Der Bootloader ist ein kleines Programm, das in einem bestimmten Speicherbereich des USB4all gebrannt wird. Dafür benutzt man ein PIC-Programmiergerät, das in der Lage ist PIC18F2455 zu programmieren. Das kann ein Brenner5 (mit der Software P18) oder ein Brenner8 (mindestens mit der Firmware V0.5 und US-Burn V1.2) sein.
Den Bootloader gibt es im USB4all-Firmware-Paket oder auf der USBoot-Seite auf meiner Homepage.
Ich empfehle die Verwendung des Bootloader-5. Bei anderen Bootloadern ist die korrekte Funktion des 2. PWM-Kanals nicht garantiert.

Der Bootloader benötigt immer den **Microchip Custom Driver**, unabhängig von der USB4all-Version. Da der USB4all-CDC einen anderen Treiber benutzt, ist zur Aktivierung des Bootloaders wie auch zur Deaktivierung des Bootloaders ein kurzzeitiges Trennen des USB-Kabels zwischen USB4all-CDC und PC nötig. Beim USB4all-MCD kann zwischen Bootloader und Firmware gewechselt werden, wärend der USB4all-MCD am PC angeschlossen ist.

Im folgenden gehe ich davon aus, dass der Bootloader sich im Steuer-PIC befindet.

9.1 Start des Bootloaders

Für den normalen Betrieb des USB4all wird der Bootloader nicht benötigt. Er bleibt inaktiv. Soll aber eine neue Firmware in den USB4all geladen werden, muss man den Bootloader „wecken“. Dafür gibt es zwei Möglichkeiten

- Aktivieren des Bootloaders per Software (nur USB4all-MCD)
- Aktivieren des Bootloaders mit dem Jumper JP1

9.1.1 Aktivieren des Bootloaders per Software

Um den Bootloader per Software zu aktivieren, beschreibt man die Zelle 0xFE des USB4all-EEPROMs mit dem Wert 0xFF. Dann löst man ein Reset des USB4all aus, oder man trennt ihn für ein paar Sekunden vom USB-Bus.

Beim **USB4all-CDC** kann man den EEPROM z.B. von einem beliebigen Terminalprogramm aus entsprechend beschreiben:

USB4all-CDC:

```
Comport1.WriteStr('#5A-2-FE-FF'+chr(0));      // EEPROM (0xFE)=0xFF
Comport1.ReadStr(rxstr, 50);                  // Quittung
```

Danach ist der USB4all-CDC vom PC zu trennen und wieder anzustecken (Reset per Firmware funktioniert hier nicht.)

Für den **USB4all-MCD** kann das die Windows-Software USBoot (ab Version 3.0)

automatisch erledigen. Findet es einen USB4all-MCD, dann kann der "**Activate Bootloader**"-Button benutzt werden, um den Bootloader des USB4all zu aktivieren und den USB4all neu zu starten.

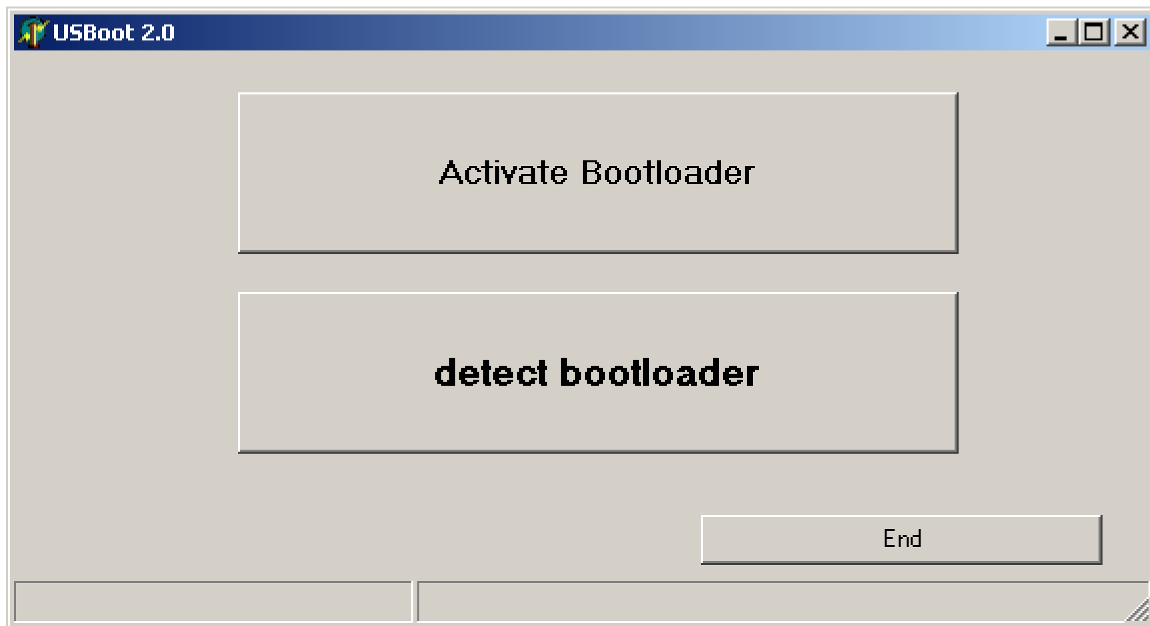


Abbildung 35 USBoot möchte den Bootloader aktivieren

Beim Neustart des USB4all-MCD wird dann der Bootloader aktiv. Danach wird dann wie üblich "**detect Bootloader**" gedrückt, um den aktivierten Bootloader zu benutzen.

Nun kann die neue Software mit USBoot in den PIC geladen werden.

9.1.2 Aktivieren des Bootloaders mit dem Jumper JP1

Es gibt auch eine Hardware-Lösung.

Der USB4all ist vom PC zu trennen (USB-Kabel trennen). Dann ist der Jumper JP1 zu stecken. Falls man den Jumper beim Bau weggelassen hat, dann verbindet man das Pin 1 des USB4all z.B. mit einem Stück Draht mit Masse (Vss).

Nun wird der USB4all wieder mit dem PC verbunden. Der Bootloader startet.
Von nun an wird der Jumper (bzw. die Masseverbindung am Pin 1) nicht mehr benötigt.

9.1.3 Neue Firmware in den USB4all laden

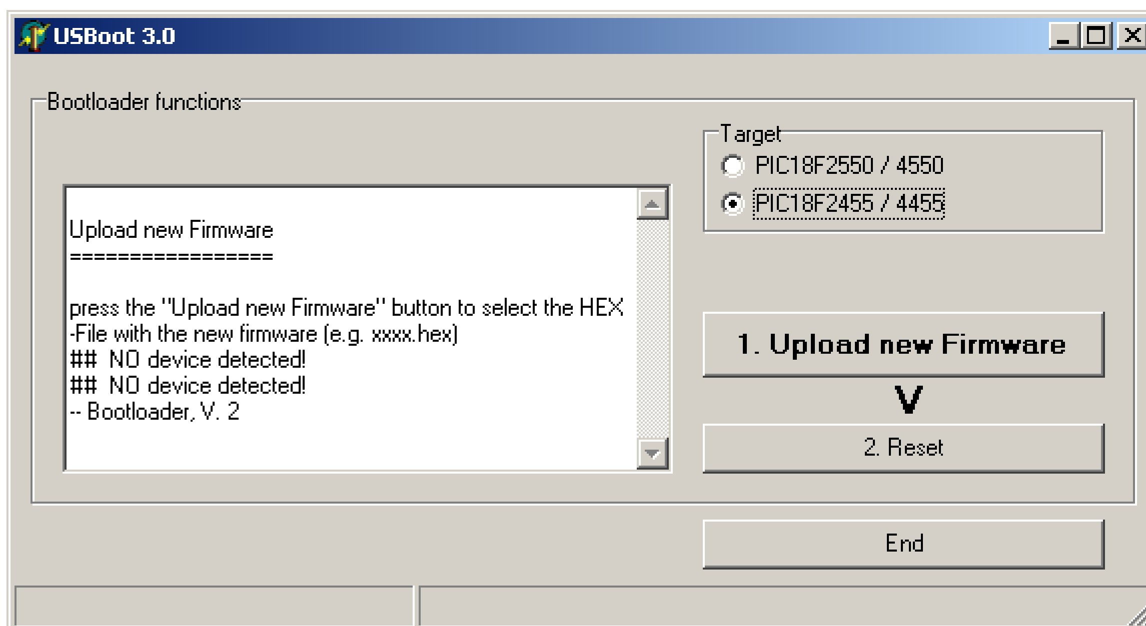


Abbildung 36 Neue Firmware in den USB4all laden

Hat man den Bootloader aktiviert und USBoot gestartet, zeigt USBoot daraufhin das obige Fenster. Im oberen rechten Auswahlfenster wählt man als Target “PIC18F2455”.

Der nächste Schritt ist das Klicken auf die Schaltfläche **1. Uplade new Firmware**. Daraufhin öffnet sich ein Auswahlfenster, in dem man das HEX-File mit der neuen Firmware auswählen muss. USBoot

- lädt das HEX-File,
- brennt die neue Firmware in den Steuer-PIC,
- prüft die gebrannten Daten auf Korrektheit und
- erklärt die neue Firmware für gültig.

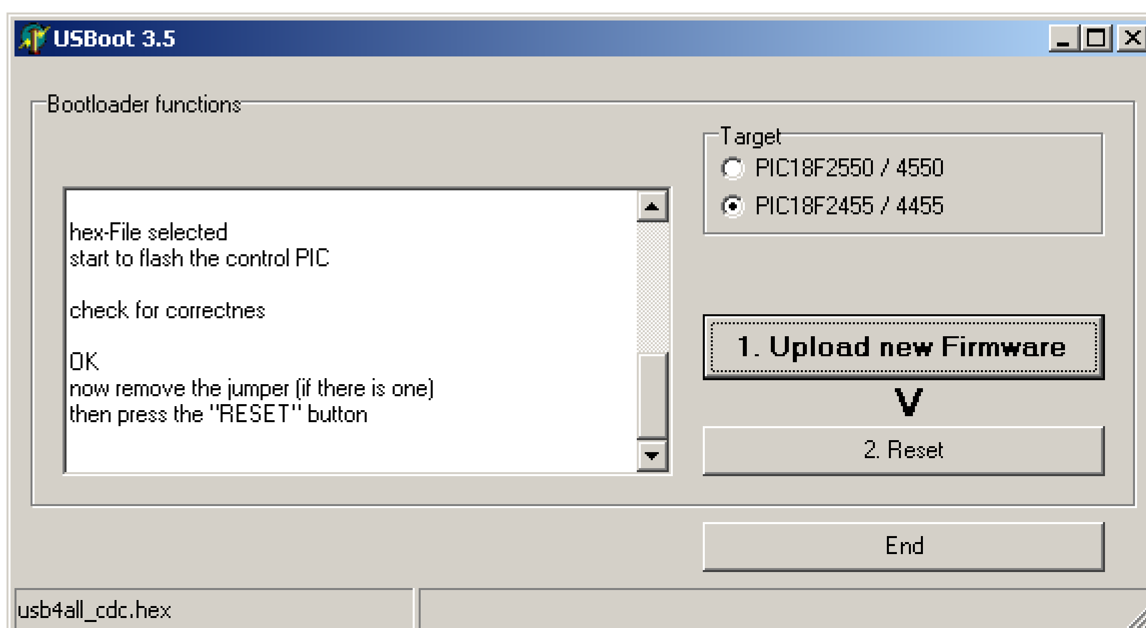


Abbildung 37 Neue Firmware geladen

Spätestens jetzt ist der Jumper JP1 zu entfernen.

Durch ein Klicken auf **2. Reset** wird der USB4all neu gestartet. Nach ca. 2 Sekunden ist er USB4all-MCD wieder betriebsbereit. Der USB4all-CDC muss dagegen vom PC getrennt werden (USB-Kabel ausstecken) und nach einigen Sekunden Wartezeit wieder mit dem PC verbunden werden.

Der Bootloader verändert NICHT die im EEPROM gespeicherten Nutzerdaten (EEPROM-Bereich 0x00..0xBF).

9.1.4 Falsches HEX-File in den USB4all geladen

Der Bootloader überprüft nicht, ob es sich bei dem Hex-File wirklich um eine USB4all-Firmware handelt.
Falls man versehentlich ein falsches HEX-File ausgewählt und per Bootloader in den PIC geladen hat, ist nichts verloren (außer der alten Firmware natürlich). Der Bootloader verändert weder die Konfiguration des Steuer-PIC noch die im EEPROM gespeicherten Nutzerdaten. Der Bootloader kann sich nicht selbst zerstören.

Der Bootloader kann in jedem Fall immer noch mit dem Jumper JP1 aktiviert werden, und dann kann man das richtige HEX-File in den Steuer-PIC brennen.

10 Fehlersuche bei USB-Geräten

10.1 Allgemein

Der erste Schritt ist immer der schwierigste.
Obwohl die Hardware der USB-Schnittstelle eigentlich recht problemlos ist, kommt es doch manchmal vor, dass man sein frisch zusammengebautes USB-Gerät an den PC anschließt, und es vom PC nicht korrekt erkannt oder behandelt wird. In diesem Moment steht der Bastler dann meist etwas hilflos da. Ich möchte hier eine Hilfestellung geben, um das "störrische" USB-Device in Betrieb nehemen zu können:

10.2 Treiber und Device (Windows)

USB-Geräte sind plug&play-Geräte, das Betriebssystem wählt also automatisch den zum Gerät passenden Treiber aus. Woher weiss z.B. Windows, welcher Treiber der richtige ist?

Aus der inf-Datei des Treibers. Um einen Treiber zu installieren, wird eine inf-Datei verwendet. In dieser stehen alle möglichen Informationen über diesen Treiber. Unter anderem steht da auch für welche USB-Devices Windows diesen Treiber verwenden soll.

Eindeutig erkannt wird ein Device an seiner VID-PID-Information. Das ist eine Kombination aus zwei Zahlen. Die erste Zahl (VID) ist die Herstellerkennung (vendor-ID) die zweite Zahl ist die vom Hersteller für dieses Gerät vergebene Geräte-Kennung (product-ID).

In der inf-Datei steht also, für welche VID-PID-Kombinationen der Treiber verwendet werden soll.

In der inf-Datei für mein USB4all findet man z.B. folgenden Eintrag:

```
[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_FF0B
```

Der mit der inf-Datei mitgelieferte Treiber soll also verwendet werden, wenn ein Gerät mit der VID0x04D8 und der PID 0xFF0B gefunden wird. Windows merkt sich das. Wird später einmal ein Gerät mit dieser VID-PID gefunden, dann verwendet Windows automatisch den vorgesehenen Treiber.

10.3 Anschluss am PC

Was passiert nun genau, beim Anstecken eines USB-Devices am PC?

Wenn man sich einen USB-Stecker anschaut, dann sieht man, das zwei seiner vier Kontakte etwas länger sind und deshalb beim Einstecken zuerst Kontakt bekommen. Das sind die Pins für Masse (GND) und +5V (VBUS). Das Device wird also schon mit Strom versorgt, bevor es (Sekundenbruchteile später) mit den beiden Kontakten des Datenbusses (D+, D-) an den USB-Bus angeschlossen wird. Dadurch kann es sich auf die Verbindung vorbereiten.
Beim Trennen des Steckers erkennt das Device zuerst den Kontaktverlust der Kontakte D+ und D-. Es hat dann noch ein wenig Zeit "aufzuräumen", bevor es auch der Betriebsspannung (VBUS, GND) beraubt wird. Das ist Voraussetzung für die hotplug-Fähigkeit.

Doch zurück zum Anstecken des Devices am PC.

Das Device erhält aus dem VBUS-Pin eine Betriebsspannung von ca +5V. Auf den Datenleitungen beträgt der Signalpegel aber 0V oder 3,3V. Deshalb muß das Device intern 3,3V erzeugen. Dafür ist ein Spannungsregler im Chip des PIC18Fxxxx vorhanden. Dieser benötigt einen Siebkondensator am Pin VUSB (100 ... 470 nF).

Das Device zieht nun einen der Daten-Pins auf 3,3V (Welchen Pin, das hängt von der geplanten Datenrate der Verbindung ab.) Im PIC18Fxxxx gibt es dafür interne pull-up-Widerstände.
Der PC bemerkt die 3,3V auf der Datenleitung, und weiß nun, dass sich am Anschluss ein USB-Device befindet. Es ist aber noch nicht der Typ des Devices bekannt.
Vorübergehend wird das Gerät nun als unbekanntes USB-Device bezeichnet.

Als nächstes nimmt der PC mit dem neuen Device Kontakt auf. Dabei werden über die Datenleitungen erstmals serielle Daten ausgetauscht. Dabei wird nun die VID-PID ausgelesen. Windows vergleicht diese mit allen ihm bekannten VID-PIDs. Ist die VID-PID unbekannt, dann fordert es zur Treiberinstallation auf. Wurde der Treiber aber schon vorab installiert, dann ist Windows die VID-PID bekannt, und der Treiber wird geladen. Von nun an erscheint das Gerät im Gerätemanager in seiner richtigen Geräteklasse.

Der Jumper JP1 dient als Backup zur Aktivierung des Bootloaders.

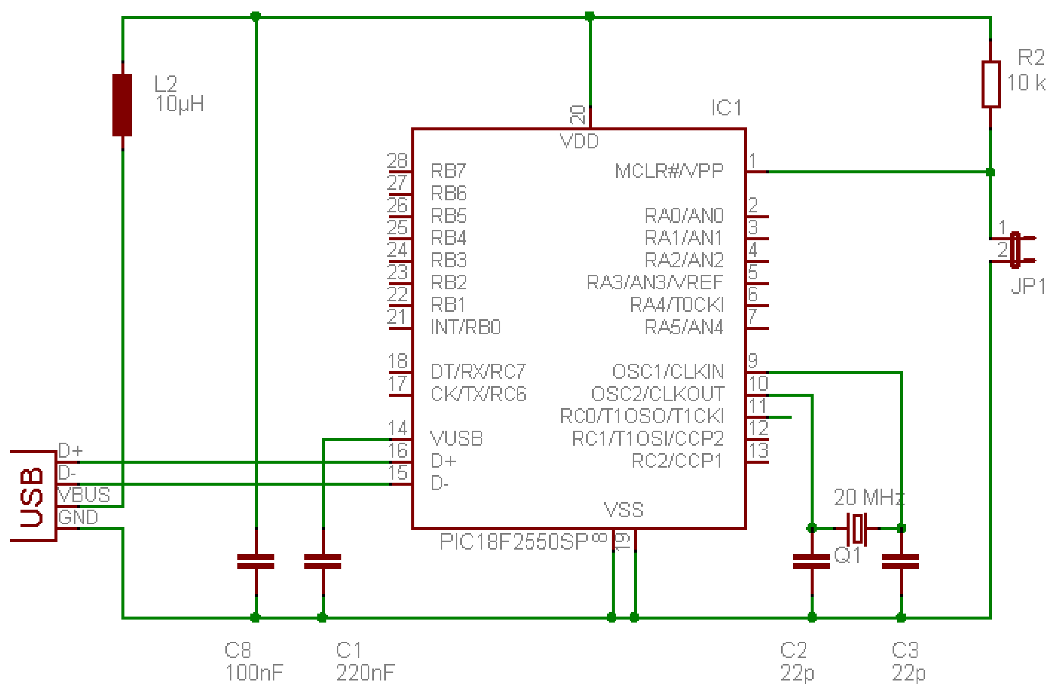


Abbildung 38 USB-Grundbeschaltung eines PIC

10.4 Typische Fehlerbilder

Der PC reagier nicht auf das Anstecken des Device

Oops. In diesem Fall werden nicht einmal die 3,3V vom Device auf die Datenleitung gelegt. Es könnte sein, dass der PIC keine Spannung erhält. Deshalb sollte man zuerst die Spannung zwischen Vdd und Vss des PIC messen. Sie sollte im Bereich von 4,5V..5,5V liegen. Fehlt sie, dann hat man irgendetwas nicht richtig angeschlossen.

Ist die Vdd-Vss-Spannung in Ordnung, dann prüft man als Nächstes die Spannung über dem VUSB-Kondensator (der Kondensator zwischen dem VUSB-Pin des PIC und Vss/GND/Masse). Sie muss etwa 3,3V betragen. Fehlt die Spannung, dann ist der PIC warscheinlich nicht korrekt programmiert worden.

Der PC erkennt das Device als unbekanntes USB-Gerät

Der PC erkennt die 3,3V auf der Datenleitung, kann aber die VID-PID nicht aus dem Device auslesen.
Das liegt häufig an einem falschen USB-Takt des Device. Da dieser Takt aus dem Quarztakt des PIC18Fxxxx gewonnen wird, könnte ein falscher Quarz eingesetzt worden sein. Wurde z.B. anstelle eines vorgesehenen 20-MHz-Quarzes eine 18,3-MHz-Quarz einsetzt, dann wird man mit Sicherheit dieses Fehlerbild erhalten.
Eine weitere Fehlerursache war hier schon das Vertauschen der beiden Leitungen D+ und D-.

Das Device wird korrekt erkannt, aber bei längeren Transaktionen gibt es Fehlermeldungen

Das passiert typischerweise, wenn der 3,3V-Spannungsregler im PIC18Fxxxx nicht richtig arbeitet, dann bricht der Daten-Spannungspegel bei langen Transfers zusammen.
Die Ursache ist typischerweise der Kondensator zwischen Vss (GNG, Masse) und dem VUSB-Pin des PIC. Die genaue Größe des Kondensators ist zwar unkritisch, aber er muss vorhanden und richtig angelötet sein.

Eine weitere Möglichkeit ist ein Timeout. Der PC gibt dem Device eine Aufgabe und wartet dann eine gewisse Zeit auf eine entsprechende Antwort des Device. Auch die USB4all-Firmware reagiert auf jede PC-Aktion mit einer Quittung. Erfolgt aber die Reaktion des Devices nicht innerhalb einer begrenzten Zeit (z.B. 1 Sekunde), führt das zu einem Timeout-Fehler. Die Firmware des USB4all sollte aber eigentlich immer schnell genug reagieren. Der Frequenzmesser kann schon mal 100 ms benötigen, und die Servosteuerung 33 ms, beides ist aber noch unkritisch.
Problematisch ist nur der synchone Betrieb von Schrittmotoren. Der ist also die Ausnahme, und sollte nur eingesetzt werden, wenn die Motordrehung in wenigen Millisekunden abgeschlossen ist. Ansonsten verwendet man besser den asynchronen Betrieb.