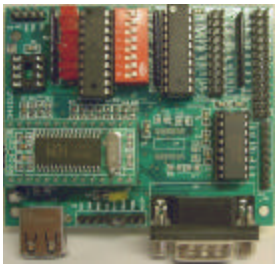
Universeller USB-Interface-Chip CH341A

Die Lösung für Ihr USB-Hardware-Projekt!

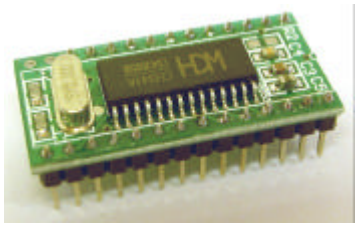
USB-RS232-Adapter
USB-Printer-Adapter
EPP/MEM Read/Write
Digital I/O
I²C-Interface

Der CH341A erlaubt den Betrieb verschiedenster Anwenderschaltungen am USB, ohne dass dazu zwingend die Verwendung (und Programmierung) eines Mikrocontrollers notwendig wäre. So lassen sich z.B. per I²C-Interface AD/DA-Wandler, Port-I/Os, usw. anschliessen. Ein 8-Bit-Datenbus erlaubt parallele Datenübertragung per EPP und MEM und macht so z.B. den direkten Anschluss und Betrieb von HD44780 kompatiblen Text-Displays am USB möglich. Auch bestehende Anwendungen, die für RS-232 konzipiert wurden (z.B. mit AVR), können schnell auf den USB umgesetzt werden. Darüber hinaus bietet die Spannungsversorgung per USB oft erhebliche Vorteile. Für die Programmierung steht eine API-DLL zur Verfügung, die sich aus nahezu jeder Programmiersprache (Delphi, C, VB, ...) ansprechen lässt. Ausserdem steht ein virtueller COM-Port-Treiber für den RS232-Betrieb zur Verfügung. Darüber hinaus wird der Chip auch von unserer Software ProfiLab Expert 4.0 unterstützt.

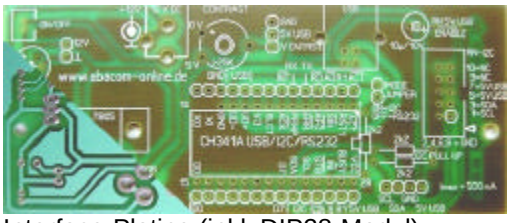
Für den schnellen und erfolgreichen Einsatz des CH341A bieten wir Ihnen folgende Komponenten an:



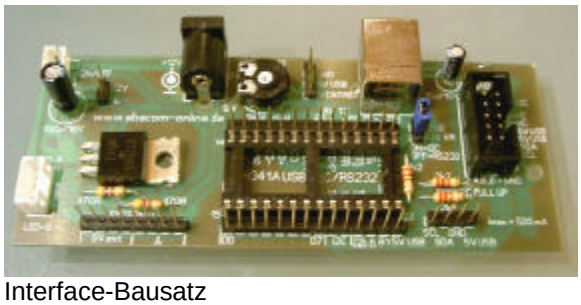
Evaluation-Board,
anschlussfertig
(inkl. DIP28-Modul)



DIP28-Modul, einbaufertig



Interface-Platine (inkl. DIP28-Modul)

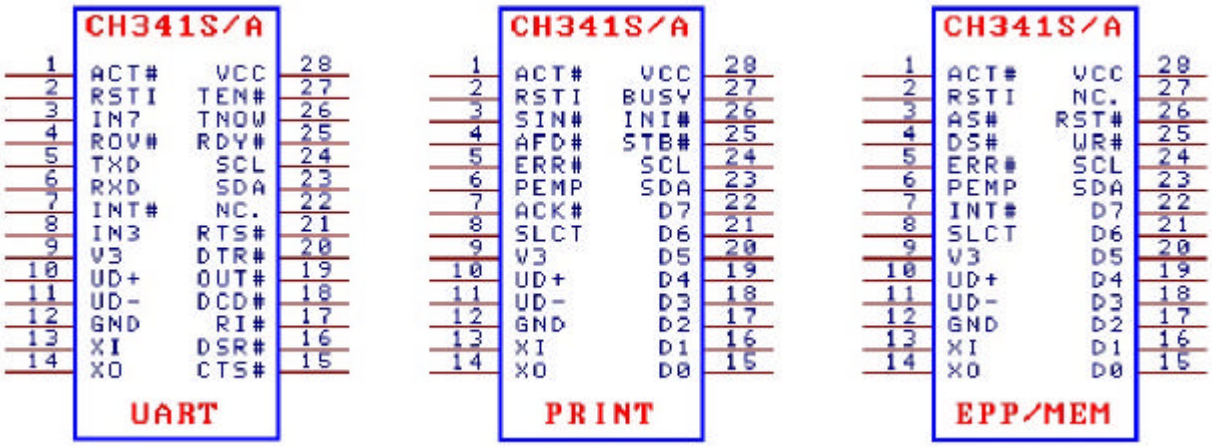


Interface-Bausatz
(inkl. DIP28-Modul, Platine und Bauteile)

Konfiguration der Betriebsart

Der Chip ist für drei grundlegende Betriebsarten konfigurierbar: Je nach Betriebsart übernehmen die Anschlüsse verschiedene Aufgaben.

- **UART**
- **PRINT**
- **EPP / MEM / I²C / Digital I/O**



Die Auswahl der Betriebsart kann über die Signalleitungen SDA und SCL oder optional mit einem EEPROM erfolgen. In Abhängigkeit der gewählten Betriebsart verwendet der Chip verschiedene USB-PID (Product ID) und Windows-Treiber.

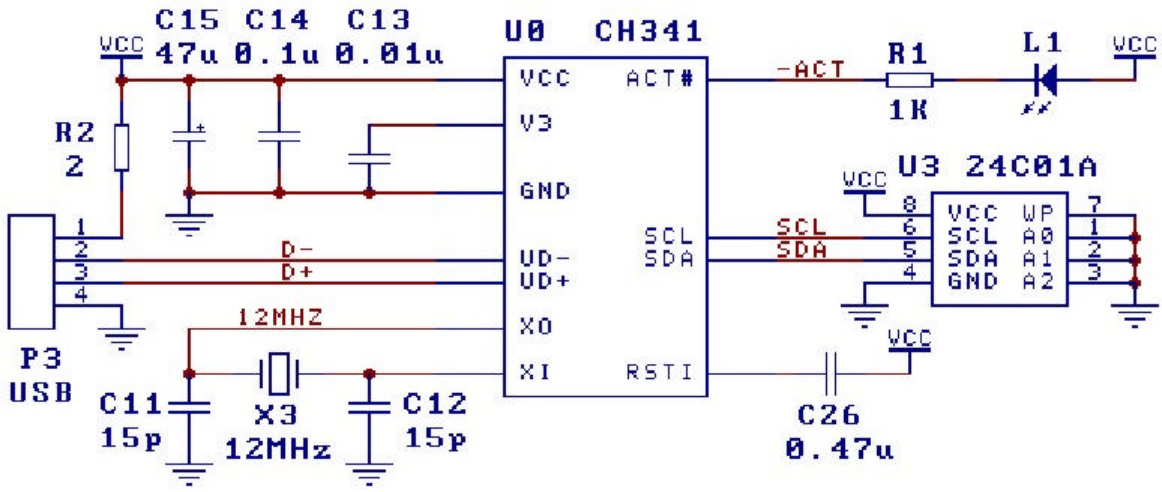
Durch den optionalen Anschluss eines EEPROM an SDA und SCL kann der Chip auch mit einer eigenen Hersteller- und Produkt-ID versehen werden. Geeignet sind z.B. 24C01A, 24C02, 24C04, 24C16, usw. Bei der Initialisierung wird zunächst die Konfiguration per EEPROM geprüft.

Schlägt die EEPROM-Erkennung fehl, erfolgt die Initialisierung entsprechend dem Zustand der SDA und SCL Leitungen. Die Betriebsart EPP/MEM kann auch erzwungen werden, idem man den Anschluss ACT# durch einen 2K-Widerstand nach Masse zieht. Dies hat den Vorteil, dass die I²C-Anschlüsse nutzbar bleiben.

SCL und SDA Konditionen	Betriebsart	USB-PID
SDA und SCL offen	Serielle Schnittstelle; RS232	5523 hex
SDA low, SCL offen	EPP/MEM/I²C/IO	5512 hex
SDA mit SCL direkt verbunden	USB-Printer	5584 hex

USB, Versorgung, Takt, Reset

Pin	Bez.	Funktion	Anmerkung
28	VCC	Versorgung	0,1 µF Entkopplung gegen Masse
12	GND	Masse	
9	V3		3,3 V: mit VCC verbinden 5 V: 0,01 µF gegen Masse
13	XI	Quarz IN	Quarz 12 MHz
14	XO	Quarz OUT	Quarz 12 MHz
10	UD+	USB D+	Direkter USB-Anschluss D+
11	UD-	USB D-	Direkter USB-Anschluss D-
1	ACT#	Ausgang	LOW nach abgeschlossener USB-Initialisierung
2	RST1	Eingang	Externer RESET-Eingang
24	SCL	I²C	SCL-Signal
23	SDA	I²C	SDA-Signal



Obiges Schaltbild stellt beispielhaft den Betrieb des CH341 am USB dar. Der elementare Anschluss ist die USB-Verbindung (P3). Die Taktung erfolgt durch ein 12 MHz-Quarz. An die von uns gelieferten Breakout-Board braucht nur noch eine USB-Buchse/-Kabel angeschlossen zu werden. Der Quarz befindet sich bereits mit auf der Adapter-Platine. Diese setzt ausserdem das SMD-Rastermass des Chips auf das bequem zu lötende DIP28-Rastermass (2,54 mm) um.

Üblicherweise haben USB-Kabel folgende Farbkennzeichnung:

VCC	**Rot**
Masse	**Schwarz**
D+	**Grün**
D-	**Weiss**

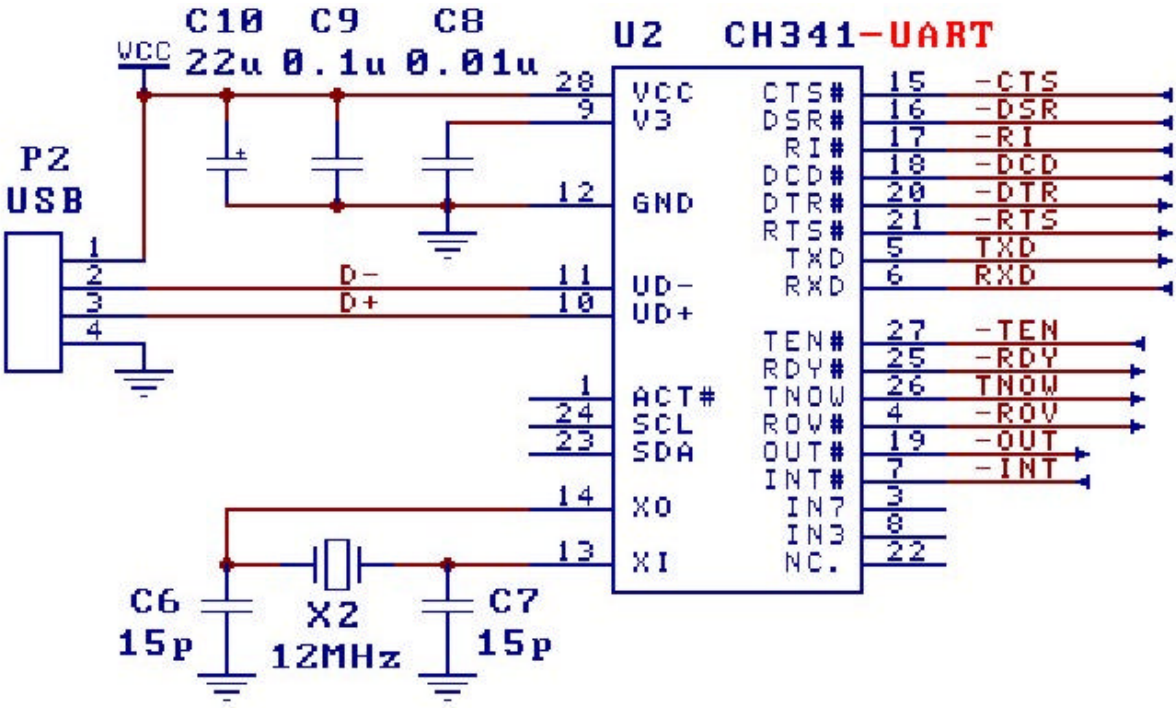
Die Verwendung eines EEPROM ist optional. Die Versorgung der Schaltung erfolgt über USB. L1 zeigt die Betriebsbereitschaft an. C26 sorgt für einen Power-On-Reset.

UART-Betrieb

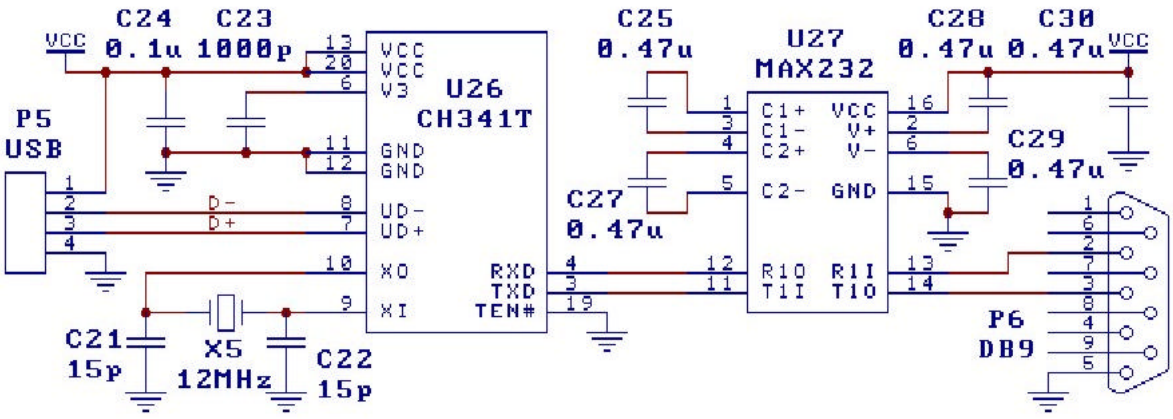
Pin	Bez.	Funktion	Anmerkung
5	TXD	Ausgang	Sendedaten
6	RXD	Eingang	Empfangsdaten
27	TEN#	Eingang	LOW = enabled; Übertragung freigeben
25	RDY#	Ausgang	LOW = Empfangsbereit
26	TNOW	Ausgang	HIGH = Übertragung in Arbeit
4	ROV#	Ausgang	LOW = Empfangspufferüberlauf
15	CTS#	Eingang	Hardware-Handshake
16	DSR#	Eingang	Hardware-Handshake
17	RI#	Eingang	Hardware-Handshake
18	DCD#	Eingang	Hardware-Handshake
20	DTR#	Ausgang	Hardware-Handshake
21	RTS#	Ausgang	Hardware-Handshake
19	OUT#	Ausgang	Reserviert
7	INT#	Eingang	Interrupt-Anforderung mit steigender Flanke
8	IN3	Eingang	Reserviert
3	IN7	Eingang	Reserviert
22	NC.	Reserviert	Reserviert

Die offenen Leitungen ACT#, SCL und SDA konfigurieren den Chip für die UART-Betriebsart.

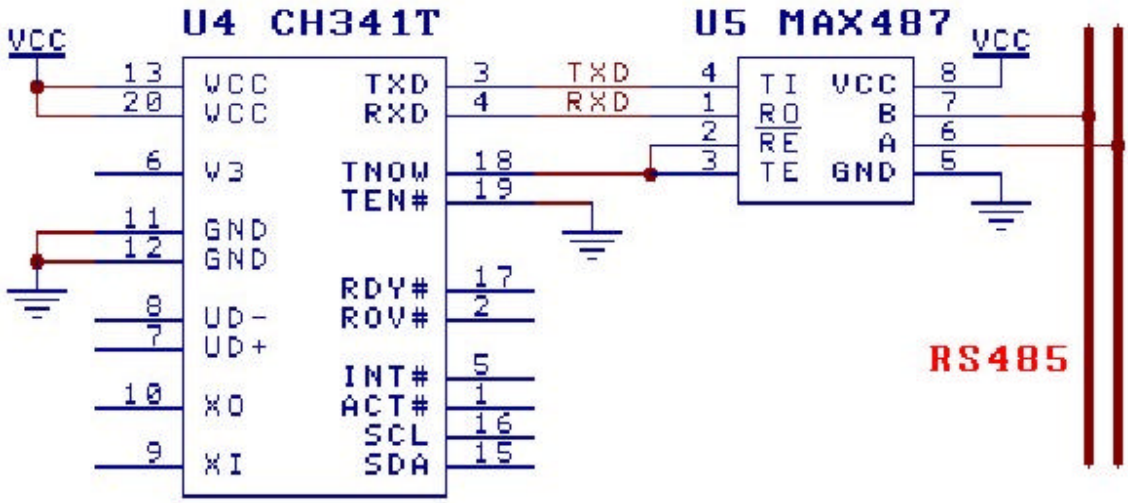
Der mitgelieferte Windows-Treiber installiert in dieser Konfiguration einen COM-Anschluss.



Die Standard-RS232-Signale stehen mit TTL-Pegel zur Verfügung und können so z.B. direkt mit Mikrocontrollern (AVR, PIC, ...) verbunden werden. Bei Bedarf können die Signale mit entsprechenden Pegelwandlern (MAX232) auf RS232-Pegel umgesetzt werden.



Ebenso ist auch der Anschluss eines RS485-Wandlers denkbar.



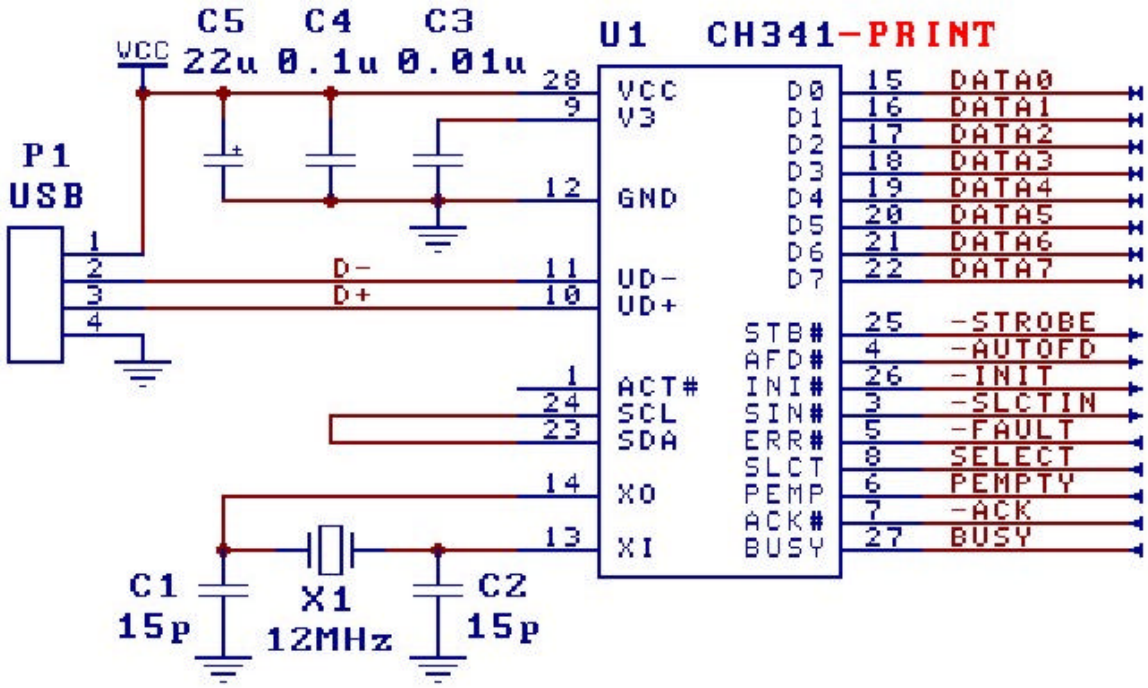
<u>Unterstützte Schnittstellen-Parameter</u>:

Datenbits:	5...9
Stopbits:	1, 2
Parität:	keine, gerade, ungerade
Baudraten:	50, 75, 100, 110, 134.5, 150, 300, 600, 900, 1200, 1800, 2400, 3600, 4800, 9600, 14400, 19200, 28800, 33600, 38400, 56000, 57600, 76800, 115200, 128000, 153600, 230400, 460800, 921600, 1500000, 2000000.

PRINT-Betrieb

Pin	Bez.	Funktion	Anmerkung
22...15	D7...D0	DATA	Parallele Datenleitungen
25	STB#	Ausgang	Strobe
4	AFD#	Ausgang	Autofeed
26	INI#	Ausgang	Init
3	SIN#		Select IN
5	ERR#	Eingang	Error
8	SLCT	Eingang	Select
6	PEMP	Eingang	Paper out
7	ACK#	Eingang	Acknowledge
27	BSY	Eingang	Busy

Der Betrieb als USB-Printer-Port ist denkbar einfach. **Die Brücke zwischen SCL und SDA konfiguriert den Chip für den PRINT-Betrieb.** Die Signalleitungen übernehmen die Funktionen eines Standard-Centronics-Anschluss für den Druckeranschluss. In dieser Konfiguration erkennt Windows den Chip als ”USB-Druckerunterstützung”. Diesen Modus wird man in der Praxis eher selten einsetzen.



EPP / MEM / I²C / I/O

Diese Betriebsart des Chips kann auf zwei verschiedene Arten konfiguriert werden:

Pin 23 (SDA) mit Masse verbinden (=> kein I²C möglich)

oder

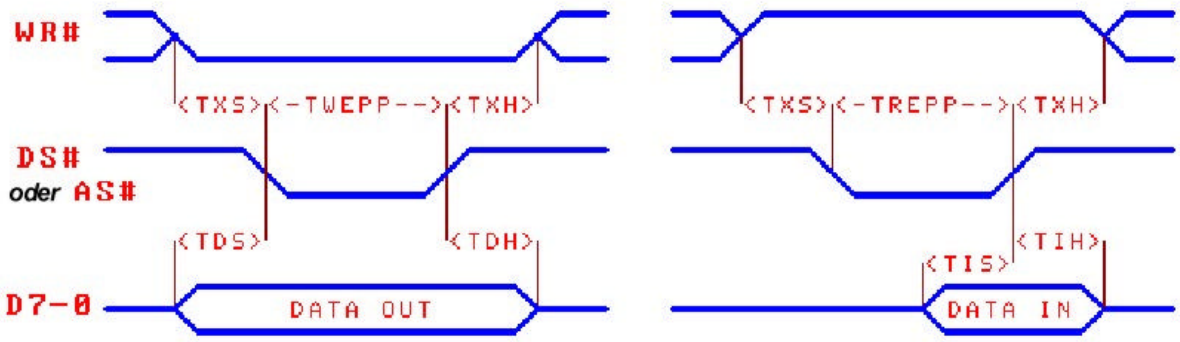
Pin 1 (ACT#) über 2K mit Masse verbinden

In dieser Konfiguration wird ein Gerätetreiber installiert, auf den mit Hilfe einer API-DLL (CH341API.DLL) aus einer Programmiersprache heraus zugegriffen werden kann. Beispiele und Include-Dateien für Delphi, VB und C sind vorhanden. Dies macht es leicht möglich eine Vielzahl von Peripherie-IC (AD/DA-Wandler, Port-IO, Speicher, etc.) am USB zu betreiben, ohne dass dazu ein Mikrocontroller notwendig wäre.

Der CH341A übernimmt dabei selbst den Datenaustausch mit der Peripherie, indem er die notwendigen Signale des jeweiligen Hardware-Protokolls an seinen Anschlusspins erzeugt. Dabei übernehmen die Anschluss-Pins je nach aufgerufener API-Funktion EPP/MEM/I²C/IO verschiedene Aufgaben. Davon unabhängig sind die Anschlüsse I²C-Anschlüsse SCL (Pin 24) und SDA (Pin 23), die den Anschlusspins fest zugeordnet sind. Darüber hinaus können alle I/O-Leitungen auch individuell angesprochen werden, so dass man weitere Hardware-Protokolle per Software simulieren kann.

EPP

Der CH341 erlaubt das Lesen (/WR=HIGH) und Schreiben (/WR=HIGH) eines 8 Bit breiten Datenbus. Wahlweise kann ein Datenregister (/DS=LOW) oder Adressregister (/AS=LOW) selektiert werden. Nachstehendes Diagramm zeigt die Signalverläufe:

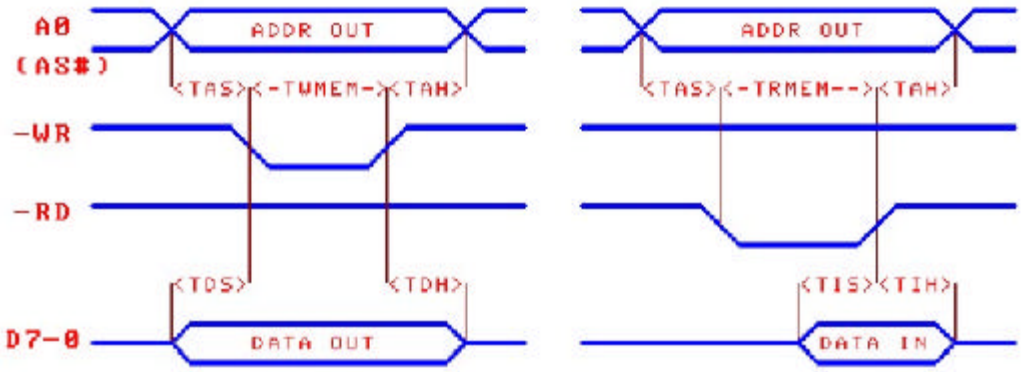


Pin	Bez.	Funktion	Anmerkung
22...15	D7...D0	Bidirektional	8 Bit Datenbus
25	WR#	Ausgang	Read/Write
4	DS#	Ausgang	/Data select
26	RST#	Ausgang	/Reset
3	AS#	Ausgang	/Adress select
27	WAIT#	Eingang	/Wait
7	INT#	Eingang	Interruptanforderung mit steigender Flanke
5	ERR#	Eingang	IN0
8	SLCT	Eingang	IN3
6	PEMP	Eingang	IN1

API-Funktionen:
Function CH341EppReadData(...) Lesen Datenregister /WR=1; /DS=0;
Function CH341EppReadAddr(...) Lesen Adressregister /WR=1; /AS=0;
Function CH341EppWriteData(...) Schreiben Datenregister /WR=0; /DS=0
Function CH341EppWriteAddr(...) Schreiben Adressregister /WR=0; /AS=0
Function CH341EppSetAddr(...) Schreiben Adressregister /WR=0; /AS=0 (1 Byte)

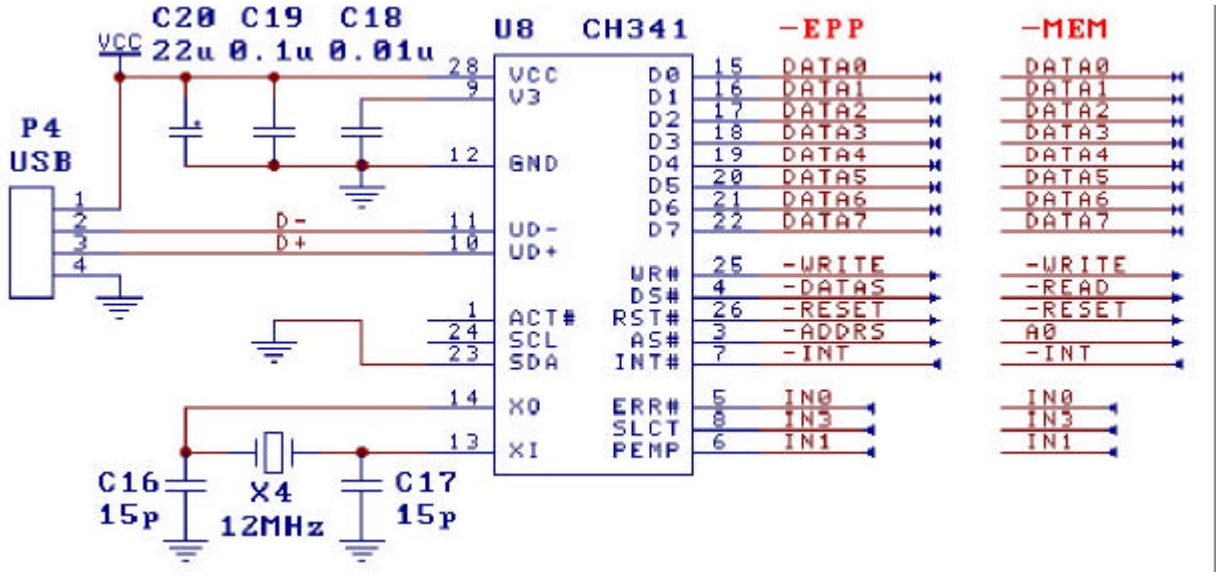
MEM

Der CH341 erlaubt das Lesen (/RD=LOW) und Schreiben (/WR=LOW) eines 8 Bit breiten Datenbus. Über die Adressleitung A0 können zwei Register adressiert werden (wie z.B. das Schreiben von DAC A und DAC B eines LTC7528 Dual 8-Bit DA Converters oder eines HD44780 kompatiblen LCD-Displays).



Pin	Bez.	Funktion	Anmerkung
22...15	D7...D0	Bidirektional	8 Bit Datenbus
25	WR#	Ausgang	/Write
4	RD#	Ausgang	/Read
26	RST#	Ausgang	/Reset
3	A0	Ausgang	Address A0
27	WAIT#	Eingang	/Wait
7	INT#	Eingang	Interruptanforderung mit steigender Flanke
5	ERR#	Eingang	IN0
8	SLCT	Eingang	IN3
6	PEMP	Eingang	IN1

API-Funktionen:
Function CH341MemReadAddr0(...) Lesen; A0=LOW; /RD=0
Function CH341MemReadAddr1(...) Lesen; A0=HIGH /RD=0
Function CH341MemWriteAddr0(...) Schreiben; A0=LOW; /WR=0
Function CH341MemWriteAddr1(...) Schreiben; A0=HIGH; /WR=0

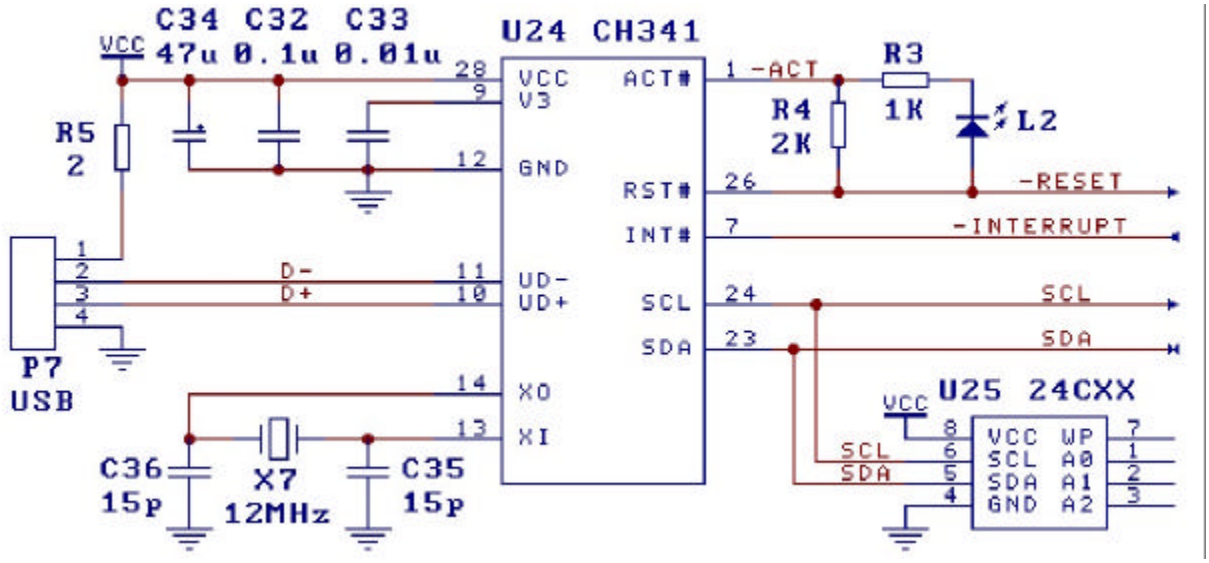


I²C (Master)

Informationen zu I²C finden Sie z.B. unter http://de.wikipedia.org/wiki/I%C2%B2C

Nachstehende Abbildung zeigt beispielhaft den I²C-Betrieb anhand eines I2C-EEPROMS. Die Busleitungen SCL und SDA erlauben den Anschluss weiterer IC mit I²C-Schnittstelle. Der CH341A unterstützt Taktraten von 20KHz/100KHz/400KHz/750KHz. Der Ch341A dient dabei als I²C-Master. Slave-Betrieb ist nicht möglich.

Da eine Chip-Konfiguration mit SDA nach Masse bei Verwendung von I²C nicht möglich ist, erfolgt die Chip-Konfiguration hier mit dem Widerstand R4. Der Anschluss RST# zieht den Anschluss ACT# bei einem Reset nach Masse und konfiguriert so den Chip-Modus. Wenn die LED L2 nicht benötigt wird, kann ACT# auch über einen 2K-Widerstand mit Masse verbunden werden und R3 entfallen.



API-Funktionen
Function CH341StreamI2C(...) Allgemeines Lesen und Schreiben über I²C
Function CH341ReadI2C(...) Lesen von Chips mit interner Speicher- oder Registeradresse
Function CH341WriteI2C(...) Schreiben von Chips mit interner Speicher- oder Registeradresse
Function CH341ReadEEPROM(...) Lesen von I²C-EEPROM
Function CH341WriteEEPROM(...) Schreiben in I²-EEPROM

Für den schnellen Einstieg in die I²C-Technik bieten wir ein I²C-Testboard von Mikrochip mit 12-Bit AD, 10-Bit-DA, 8-Bit Port-Expander, EEPROM und Temperatur-Sensor an.

SPI-Master (Funktion nicht garantiert, nur zu Information)

Pin	Bez.	Funktion	Anmerkung
22	DIN	Eingang	Data IN
21	DIN2	Eingang	Data IN2
20	DOUT	Ausgang	Data OUT
19	DOUT2	Ausgang	Data OUT2
18	DCK	Ausgang	Data Clock
17	CS2	Ausgang	CS2
16	CS1	Ausgang	CS1
15	CS0	Ausgang	CS0
24	SCL	Ausgang	SCL
23	SDA	Ausgang	SDA
26	RST#	Ausgang	Reset
7	INT#	Eingang	Interrupanforderung mit steigender Flanke
5,8,6		Eingang	Reserviert

Der Chip arbeitet als SPI-Master. Slave-Betrieb ist nicht möglich. Der Datenaustausch erfolgt über einen Datenpuffer, der vor Aufruf der entsprechenden API-Funktion mit den zu schreibenden Daten gefüllt wird. Nach dem API-Aufruf enthält der Puffer die gelesenen Daten. Es wird also die gleiche Anzahl von Bytes gelesen, die auch geschrieben wird.

Standard-SPI (MODE 0)
API-Funktion: CH341StreamSPI4
Standard-SPI (Mode 0) mit zwei unidirektionalen Datenleitungen, Clock und Chip-Select.z.B. Atmel, etc. Diese ist die gebräuchlichste SPI. Leider ist die Bezeichnung der Signale oft uneinheitlich. Der Datenaustausch erfolgt über den Eingang Pin 22 (DIN; DATA IN ; MISO) und den Ausgang Pin 20 (DOUT; DATA OUT; MOSI). Pin 18 (DCK; DATA CLOCK; SCKL) stellt den Taktausgang dar. Die Ausgänge CSx (Chip Select; Slave Select) dienen der Auswahl eines Chips. Die serielle Übertragung eines Datenbyte kann mit dem höchstwertigen Bit (MSB) oder niederwertigsten Bit (LSB) beginnen.
Eine SPI-Beschreibung findet man unter http://www.uni-koblenz.de/~physik/informatik/MCU/SPI.pdf
3-Wire SPI
API-Funktion: CH341StreamSPI3
Hier wird auf eine Datenleitung der Standard-SPI verzichtet. Dafür arbeitet die verbleibende Datenleitung hier bidirektional. Das Schreiben erfolgt bei steigender Taktflanke, das Lesen bei fallender Taktflanke. Diese Art der Datenübertragung findet man häufig bei Chips des Herstellers MAXIM, wie z.B. beim z.B. DS1626.
SPI mit doppelten Datenleitungen
API-Funktion: CH341StreamSPI5
Bei dieser Art der SPI sind die unidirektionalen Datenleitungen doppelt ausgeführt. Dabei wird ein zu übertragendes Byte in zwei Halbbytes (Nibbles) zerlegt. Das High-Nibble wird mit DIN bzw. DOUT übertragen, das Low-Nibble mit DIN2 bzw. DOUT2. Dieses Verfahren ist ungebräuchlich. Ein Beispiel für diese Art der Datenübertragung konnte nicht gefunden werden.
Andere (SPI ähnliche) serielle Datenübertragung
API-Funktion: CH341BitStreamSPI
Mit dieser Funktion lässt sich ein individuelles serielles Signalmuster generieren. So lässt sich z.B. auch ein LTC1257 DA-Wandler ansprechen, der eine 12-Bit-Datenübertragung (mit abschliessendem LOAD-Impuls im letzten Takt) verwendet. Dazu stehen 6 serielle Ausgangskanäle und zwei serielle Eingangskanäle zur Verfügung. Diese insgesamt acht Kanäle sind den Bits 0...7 der eingehenden und ausgehenden Datenbytes fest zugeordnet. Bit 6 und Bit 7 liefern nach dem API-Aufruf die Daten der eingehenden Signale D6 und D7. Die Signalleitung D3 gibt den seriellen Takt aus, der vom CH341A automatisch generiert wird. Die übrigen Datenleitungen D0...D2, D4 und D5 arbeiten als Ausgang. Der gewünschte Signalverlauf (Pattern) der Ausgangskanäle muss vor dem Aufruf der API-Funktion in die zugehörigen Bits 0..2, 4 und 5 der ausgehenden Bytes eingesetzt werden. (Die Datenrichtung der Leitungen D0 bis D5 ist zuvor noch einmalig mit der entsprechenden API-Funktion auf Ausgang zu schalten.)

Direktes Programmieren der I/O-Leitungen

Für die individuelle Programmierung stehen die folgenden Anschlüsse zur Verfügung. Jedem Bit in den Parametern Enable, SetDirOut und SetDataOut der API-Funktionen ist ein Pin des CH341A zugeordnet.

Fünfzehn bidirektionale Pins (mit programmierbarer Datenrichtung; Eingang oder Ausgang):

Bit 0-7	D0-D7	pin 15-22
Bit 8	ERR#	pin 5
Bit 9	PEMP	pin 6
Bit 10	ACK	pin 7
Bit 11	SLCT	pin 8
Bit 12	-	-
Bit 13	BUSY/WAIT#	pin 27
Bit 14	AUTOFD#/DATAS#	pin 4
Bit 15	SLCTIN#/ADDRS#	pin 3

Ein quasi-bidirektionaler (Kombination aus Eingang und Open-Collector-Ausgang) :

Bit 23	SDA	pin 23 (bei Funktionen GetInput/GetStatus)
Bit 16	SDA	pin 23 (bei Funktion SetOutput)

Drei uni-direktionale Pins (immer Ausgang):

Bit 16	RESET#	pin 26
Bit 17	WRITE#	pin 25
Bit 18	SCL pin	pin 24

API-Funktionen

CH341SetOutput
- Setzen der Datenrichtung (Eingang/Ausgang) der bi-direktionalen Pins,
- Setzen Ausgängszustände HIGH/LOW

CH341Set_D5_D0
- Setzen der Datenrichtung der Pins D0...D5
- Setzen der Ausgangszustände der Pins D0...D5

CH341GetInput und CH341GetStatus
- Lesen der Eingangszustände

Parameter Enable
- Sperrt bzw. erlaubt die Umschaltung der Datenrichtung
- Bit gesetzt = Änderung zugelassen
- Bit gelöscht = Änderungen werden ignoriert

Parameter SetDirOut:
- Umschaltung der Datenrichtung: Eingang oder Ausgang
- Bit gesetzt = Pin arbeitet als Ausgang (Vorsicht!)
- Bit gelöscht = Pin arbeitet als ist Eingang

Parameter SetDataOut
- Ausgänge schalten
- Bit gesetzt = Ausgang HIGH schalten
- Bit gelöscht = Ausgang LOW schalten

Beispiel: CH341SetOutput(0, $FF, $FF, $F0) setzt die Richtung der Datenleitungen D0...D7 auf Ausgang. Alle anderen Leitungen bleiben unverändert. D0...3 geben LOW (0 Volt) aus. D4...D7 geben HIGH aus (5 Volt).

Inhalt eines optionalen Konfigurations-EEPROM

Byte Adresse	Bez.	Erklärung	Wert
00 hex	SIG	53H = Signatur EEPROM gültig	53 hex
01 hex	MODE	23H = RS232 oder 12H = EPP/MEM/PRINTER	23 hex oder 12 hex
02 hex	CFG	**Konfiguration, siehe nächste Tabelle**	FE hex
03 hex		Reserviert	00 hex
05…04 hex	VID	Vendor ID	4348 hex
07…06 hex	PID	Product ID	55xx hex
09…08 hex	RID	Release ID	0100 hex
17…10 hex	SN	Serial Number, acht Zeichen	12345678
7F…20 hex	PIDS	Product name sting	00 hex, 00 hex
Andere Adressen			00 hex oder FF hex

Bit Nr.	Bez.	Erklärung	Wert
7	PRT	0 = USB PRINTER; sonst 1	1
6	PWR	Power: 0 = extern; 1 = USB	1
5	SN-S	Serial number 0=valid; 1=invalid	1
4	PID-S	Product string 0=valid; 1=invalid	1
3	SPD	Printer data speed 0=high; 1=low	1
2	SUSP	Suspend mode 0=prohibited; 1=permitted	1
1	PROT1		1
0	PROT0		0

Programmierschnittstelle (API)

In der Betriebsart **EPP / MEM / I²C / Digital I/O** steht für den CH341A eine Programmierschnittstelle zur Verfügung, so dass der Chip mit nahezu jeder Programmiersprache ansprechbar ist. Die API-Funktionen müssen dazu aus der Datei CH341DLL.DLL importiert werden. Entsprechende Beispiele für Delphi, C, Visual Basic und ProfiLab Expert stehen zur Verfügung.

Alle API-Funktionen bekommen als ersten Parameter die Gerätenummer (iIndex) übergeben, anhand derer ein angeschlossener Chip am USB identifiziert wird. Der erste Chip verwendet Gerätenummer 0, der zweite Chip Gerätenummer 1, usw.

Allgemeine Funktionen

Function CH341OpenDevice:
Verbindung mit CH341A herstellen. Erst danach können andere Funktionen verwendet werden

Function CH341CloseDevice:
Verbindung mit CH341A beenden

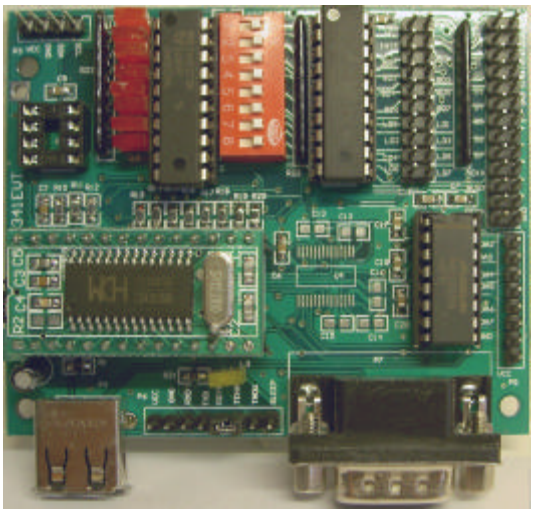
Function CH341ResetDevice:
Zurücksetzen auf Einschaltzustand

Function CH341SetExclusive:
Weiteres Öffnen verhindern.
iExclusive:
0=shared ; mehrfaches öffen möglich
1=exclusive; kein weiteres Öffnen möglich

Informationen zu den weiteren API-Funktionen finden Sie in den Beispiel-Programmen und im Anhang.

CH341A-Evaluation-Board

inkl. DIP28-Modul



Das Board demonstriert die Vielseitigkeit des CH341A und bietet bequeme Anschlussmöglichkeiten. Den vollständigen Schaltplan des CH341 EVA-Boards finden Sie in der Datei EVT_PCB.PDF.

Boardkomponenten

Adapterplatine
Der CH341-Chip ist auf einem Breakout-Board (Adapterplatine) montiert, die auf die Hauptplatine gesteckt ist (U1). Auf der Adapterplatine befinden sich bereits alle elementaren Bauteile die für den Betrieb des Chips erforderlich sind: Ein 12-MHz Quarz für die Takterzeugung und ein paar Widerstände und Kondensatoren, die den direkten Anschluss einer USB-Buchse bzw. Kabels erlauben. Alle Anschlüsse des Chips (mit Ausnahme des Quarzes) sind auf zwei Pfostenleisten geführt. Damit lässt die Adapterplatine bequem in eigenen Schaltungen einsetzen, idem man sie z.B. auf ein Experimentierboard, eine Lochrasterplatine, oder eine eigene Leitplatte steckt. Das kleine Breakout-Board ist auch einzeln erhältlich, so dass Sie Ihre eigenen Hardware-Projekte auf einfachste Weise mit USB-Funktionalität ausstatten können. Die Anschlüsse des Chips übernehmen je nach Betriebsart des Chips verschiedene Funktionen!

USB-Anschluss
Der USB-Anschluss erfolgt über ein mitgeliefertes USB-Kabel (A-Stecker-A-Stecker) und die USB-Buchse P4 die direkt mit den USB-Anschlüssen UD+ (10) und UD- (11) der Adapterplatine verbunden ist.

Serielle Schnittstelle
Der Anschluss P6 stellt die Signale seriellen Schnittstelle mit TTL-Pegel (!) zur Verfügung. Die Signale RXD und TXD sind über einen Pegelwandler (MAX232; U6) auf einen 9-poligen-D-Stecker (P7) geführt. Für die zusätzliche Pegelwandlung der Hardware-Handshake-Leitungen ist im Layout Platz für einen Pegelwandler (U4) vorgesehen. Dieser ist jedoch nicht bestückt. Falls Sie diesen Schaltungsteil selbst bestücken wollen, muss zuvor U6 aus seinem Sockel entfernt werden.
Für den RS232-UART-Betrieb sind die Anschlüsse TEN# und RDY# am Schluss P6 mit einem ein Jumper zu verbinden. Für andere Betriebsarten muss dieser Jumper entfernt werden.

Druckeranschluss
Anschluss P2 stellt die Standard-Signale eines parallelen Druckeranschlusses an einer 2-reihigen Pfostenleiste bereit. Die Pinbelegung des P2-Port entspricht dem eines 25-poligen-Sub-D-Druckeranschluss. (Entsprechende Kabel findet man häufig in ausgedienten PC´s)

Digital I/O
Der Anschluss P1 stellt an einer 20-poligen, 2-reihigen Pfostenleite acht digitale TTL-Ausgänge (LD0..LD7) und acht digitale TTL-Eingänge (BD0..BD7), sowie GND (Masse) und VCC (USB-Betriebsspannung; Vorsicht!) bereit.

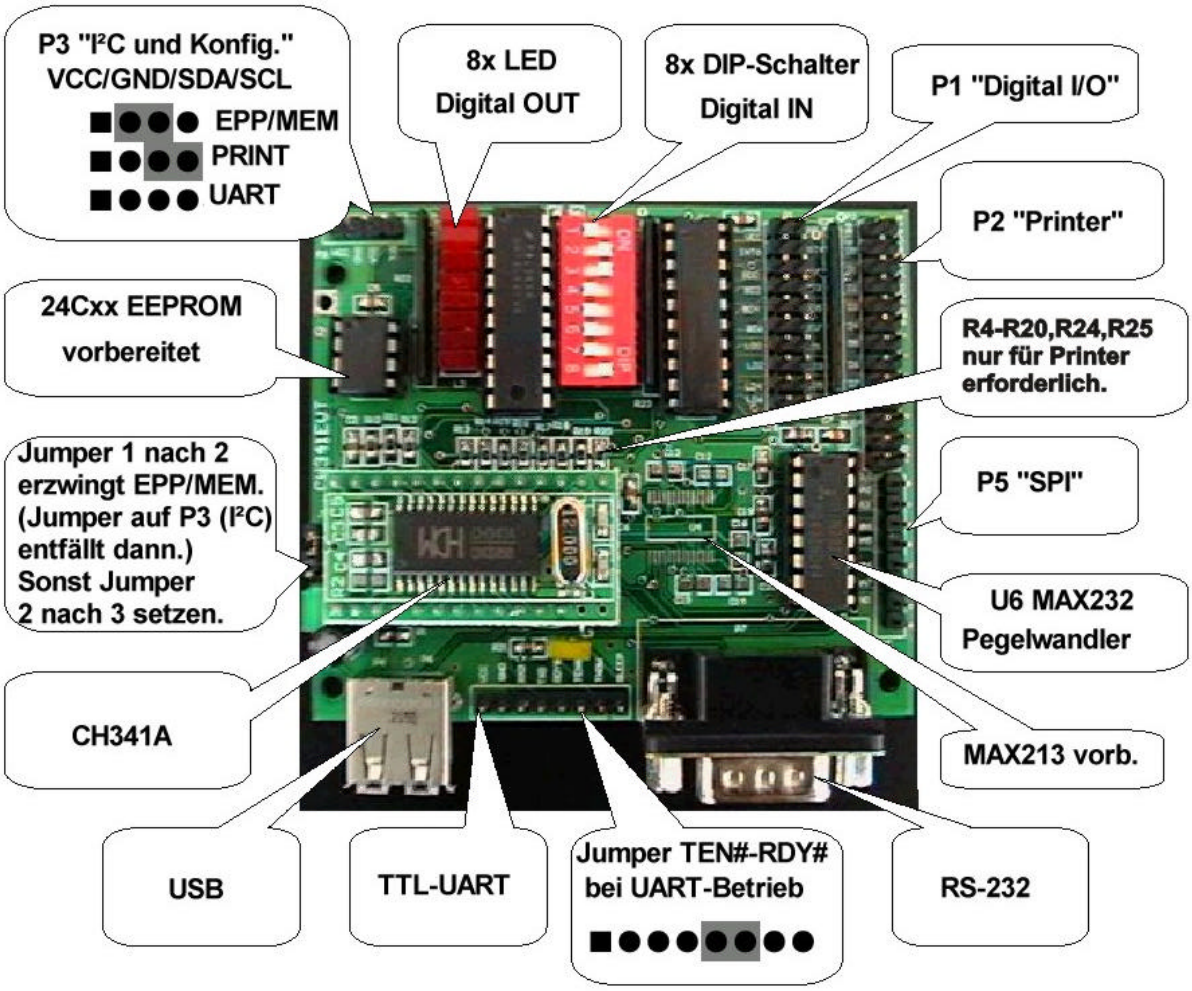
Die digitalen Ein- und Ausgänge sind über zwei gesockelte TTL-Latches (74LS274 / 74HC245) realisiert. Zu Testzwecken sind die Augänge mit LED´s beschaltet. Die Eingänge lassen mit einem 8-fach DIP-Schalter einzeln nach Masse kurzschliessen (Schalterstellung ON = Eingang LOW). Bei externer Beschaltung der Eingange müssen die Schalter unbedingt offen sein! (Schalterstellung AUS)

I²C
Anschluss P3 stellt die I²C-Signale SCL und SDA, sowie GND (Masse) und VCC (USB-Betriebsspannung; Vorsicht!) bereit und ermöglicht so den bequemen Anschluss weiterer I²C-Chips. In den IC-Sockel U5 kann ein I²C-EEPROM (24Cxx) gesteckt werden. Da man in den meisten Fällen auf ein EEPROM verzichtet, lässt sich durch Jumpern der Leitungen SDA und SCL die Betriebsart des Chips bestimmen. Konfiguration der Betriebsart durch Jumper:

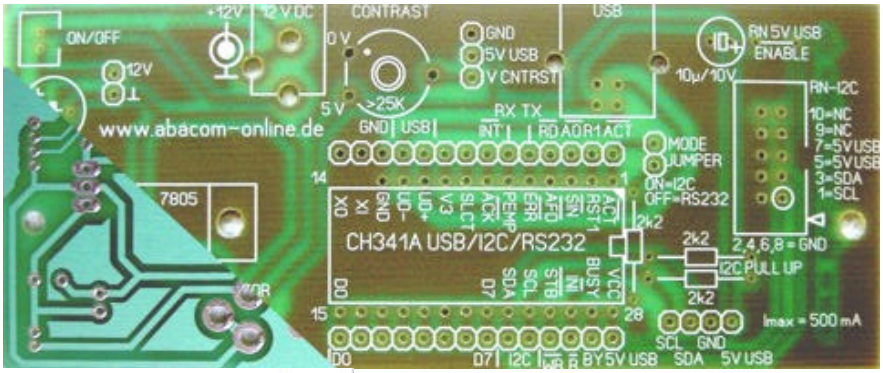
Jumper SDA<->SCL gesteckt P3(3) – P3(4): Betriebsart PRINT
Jumper nicht gesteckt P3 offen: Betriebsart: UART
Jumper SDA<->GND gesteckt P3(3) – P3(2): Betriebsart EPP/MEM/I²C/IO

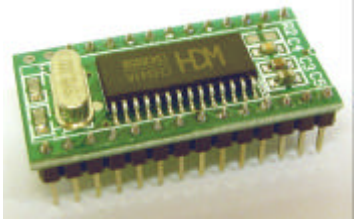
Da der I²C-Bus bei Konfiguration durch Jumper nicht mehr zur Verfügung stünde, kann die Betriebart EPP/MEM/I²C/IO auch dadurch Konfiguriert werden, dass man die Kontakte 1 und 2 am Anschluss J1 mit einem Jumper verbindet. Dadurch wird eine Verbindung zwischen Pin1 (ACT#) und Pin26 (RST#) des Chips über einen 2K-Widerstand hergestellt.

SPI (Einsetzbar nur mit SPI-fähigen Chip-Versionen)
Die SPI-Signale sowie GND (Masse) und VCC (USB-Betriebsspannung; Vorsicht!) stehen am Anschluss P5 zur Verfügung.


P3 "I²C und Konfig."
VCC/GND/SDA/SCL
EPP/MEM
PRINT
UART
8x LED
Digital OUT
8x DIP-Schalter
Digital IN
P1 "Digital I/O"
P2 "Printer"
24Cxx EEPROM
vorbereitet
R4-R20,R24,R25
nur für Printer
erforderlich.
Jumper 1 nach 2
erzwingt EPP/MEM.
(Jumper auf P3 (I²C)
entfällt dann.)
Sonst Jumper
2 nach 3 setzen.
P5 "SPI"
U6 MAX232
Pegelwandler
CH341A
MAX213 vorb.
USB
TTL-UART
Jumper TEN#-RDY#
bei UART-Betrieb
RS-232

Interface-Platine
inkl. DIP28-Modul





Die Platine wurde von uns entworfen, um Ihnen den Aufbau eigener USB-Projekte mit dem CH341A möglichst einfach zu machen:

- Leichtes Bestücken durch Bestückungsaufdruck
- Einfaches Löten durch Lötstoplack,
- Kein SMD löten erforderlich
- Individuelle Bestückungsoptionen
- Passender Einbau in ein externes 3,5”-Festplattengehäuse (FANTEC DB-337U2-B) vorgesehen
- ca. 110 x 45 mm

Die Platine ist industriell gefertigt und mit Lötstoplack und Bestückungsaufdruck versehen. Das kleinste vorkommende Rastermass ist 2 mm (kein SMD). Die Bestückung kann individuell vorgenommen werden.

Minimal-Bestückung für USB->RS232(TTL)
Dazu müssen Sie nur die USB-B-Buchse und die CH341A-Adapterplatine einlöten, sowie die beiden Anschlüsse /WR (Pin 25) und BY (Pin 27) der Platine verbinden.

Minimalbestückung für USB->I²C
Dafür muss zusätzlich ein 2K2-Widerstand und eine Drahtbrücke eingelötet werden.
Die Verbindung Pin25 – Pin27 entfällt in diesem Fall.

Optionale Bestückungsoptionen

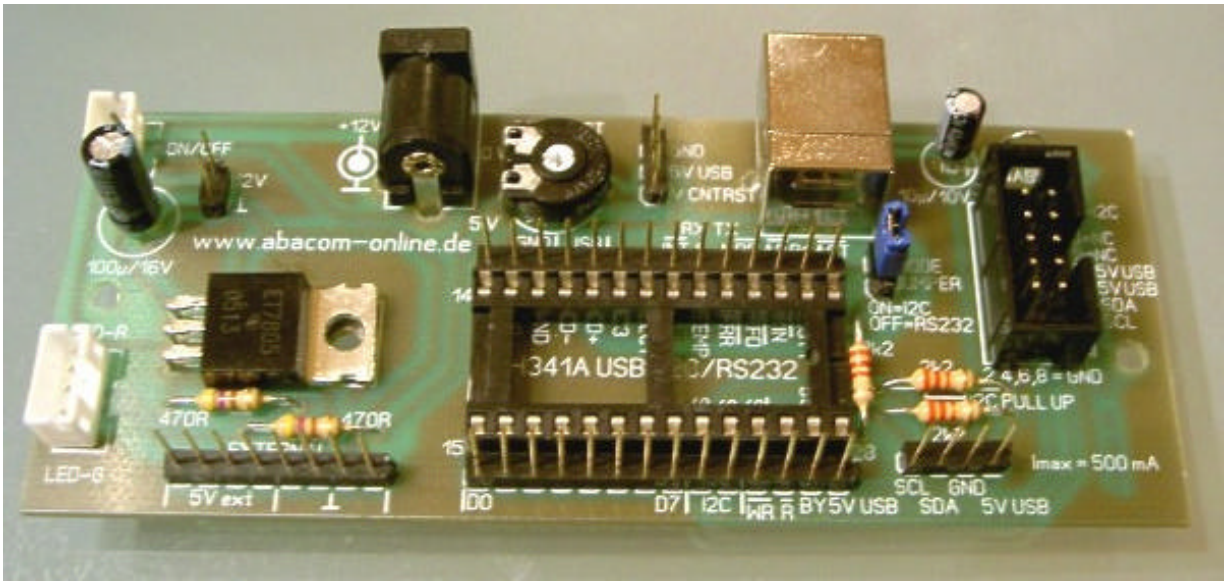
- Statt das CH341A-Moduk direkt in die Platine zu löten, empfehlen wir Buchsenleisten zu verwenden, um das Modul servicefreundlich zu sockeln.
- 2-polige Stiftleiste mit Jumper für die Konfiguration von wahlweise RS232- oder I²C-Betrieb
- Siebelko (10µ/16V) für die USB-Versorgungsspannung
- Pullups-Widerstände (2x 2K2) für die I²C-Signalleitungen SDA und SCL. Diese sind nur dann erforderlich, wenn die angeschlossene I²C-Schaltung diese nicht vorsieht.
- Vierpolige Stift- oder Buchsenleiste (5V-USB, USB-Masse, SDA, SCL), sofern man die Leitungen nicht direkt einlötet.
- 10-poliger Wannenstecker als I²C-Anschluss. Die Anschlussbelegung folgt den Empfehlungen von http://www.roboternetz.de/wissen/index.php/RN-Definitionen#I2C-Bus_Stecker . Hier findet man

auch zahlreiche Informationen und Schaltungen u.a. zum Thema I²C. Mit der Drahtbrücke”RN 5V USB ENABLE” kann bei Bedarf die USB-Versorgungsspannung auf den RN-I²C-Wannenstecker (Pin 5 und 7) aufgeschaltet werden.

- Alle Anschlüsse des CH341A (mit Ausnahme der Quarzanschlüsse XI und XO) sind auf zwei je 14-polige Lochreihen geführt. Hier lassen sich Stift- oder Buchsenleisten bestücken, sofern man die Anschlussleitungen nicht direkt einlötet.
- Eine einstellbare Hilfsspannung (VCONSTRAST = 0...5V USB) erhält man wenn man das Trimmpoti (mindestens 25K Ohm) bestückt. Diese ist auf einen dreipoligen Anschluss (GND, 5V USB, VCNTRST) geführt. Diese ist praktisch, wenn man z.B. eine LCD-Kontrast-Spannung, experimentelle Referenzspannung, einstellbare Triggerschwelle, o.ä. benötigt.
- Eine zusätzliche, externe Spannungsversorung (für die Anwenderschaltung) lässt sich realisieren, wenn man die DC-Buchse, den Siebelko (100µ / 16V) und den Spannungsregler 7805 bestückt. Für die externe Spannung ist ein Schalter ON/OFF vorgesehen. Dieser Schaltungsteil ist potentialfrei gegenüber der USB-Masse aufgebaut, d.h. die USB-Masse und die Masse der externen Versorgungsspannung sind NICHT verbunden. Die Eingangsspannung vom externen Steckernetzteil (12V; Plus innen; Minus aussen) ist nach dem ON/OFF-Schalter auf einen zweipoligen Anschluss (12V; GND ext.) geführt. Die geregelte 5V-Spannung steht am Anschluss EXTERNAL bereit, der je vier Anschlüsse (5V / GND ext.) besitzt.
- Der Anschluss LED-R / LED-G ist für den Betrieb von zwei Leuchtdioden vorgesehen, die das Vorhandensein der USB-Versorgungsspannung (LED-G) und der externen Versogungsspannung (LED-R) anzeigen. Dafür sind die beiden Vorwiderstände mit je 470R zu bestücken.

Interface-Platine

inkl. DIP28-Modul, Platine und Bauteile



Gehäuseeinbau (Gehäuse nicht im Lieferumfang)
Die Platine ist so ausgelegt, dass sie genau in ein externes 3,5”-Festplattengehäuse (FANTEC DB-337U2-B) passt. Das Gehäuse ist weit verbreitet und für ca. 15-20 Euro überall im Internet zu bekommen. Das Gehäuse wird bereits mit einem passenden, externen 12V-Netzteil geliefert. Der schraubenlose Verschluss der zwei Gehäusehalbschalen macht u.a. auch eine Wandmontage leicht möglich.

Für den Einbau wird die vorhandene Elektronik einfach ausgebaut und durch unser CH341A-Interface ersetzt. Die beiden Schrauben sollte man also gut aufheben und wieder verwenden. Im Gehäuse sind bereits der Ein-/Ausschalter und die beiden Leuchtdioden mit Steckanschlüssen vorhanden. Wer möchte kann mit etwas Geschick die DC-Buchse, die USB-Buchse und die beiden Mini-Wannenstecker für Schalter und LED´s aus der ausgebauten Elektronik auslöten und für sein USB-Interface wieder verwenden. Nach dem Einbau der Elektronik bietet das Gehäuse noch genügend Platz um eine 100x160 mm-Standardplatine mit einer eigenen Anwenderschaltung bequem aufzunehmen. Auch der Einbau von weiteren Anschlüssen, Anzeigen und Bedienungselementen in das Kunststoffgehäuse mit Alu-Finish ist recht einfach.

Für alle, die der aufgedruckte Schriftzug ”Mobile Disk” stört abschliessend noch ein Tipp: Mit einer flach aufgesetzten Rasierklinge erst die erhabenen Teile der Druckschicht vorsichtig abspachteln und dann mit einem Topfschwamm die Reste in Richtung der Alu-Maserung wegpolieren. Bei unserem Labormuster gelang das rückstandslos und ohne jede Beschädigung. (Lösungsmittel erwiesen sich hingegen als ungeeignet!)

Anhang: API-Funktionen (CH341DLL.DLL)

```
// **************
// USB Funktionen
// **************
type    mPCH341_NOTIFY_ROUTINE =ProceDure (iEventStatus:cardinal); stdcall;
const   CH341_DEVICE_ARRIVAL =        3 ;
        CH341_DEVICE_REMOVE_PEND =    1 ;
        CH341_DEVICE_REMOVE    =      0 ;
Function CH341SetDeviceNotify(iIndex:cardinal; iDeviceID: PCHAR;
iNotifyRoutine:mPCH341_NOTIFY_ROUTINE):boolean;Stdcall; external 'CH341DLL.DLL';


// *********************
// Allgemeine Funktionen
// *********************

// Alle Funktionen addressieren einen CH341A am USB mit dem Parameter 'iIndex':
// iIndex=0 => 1. CH341 am USB
// iIndex=1 => 2. CH341 am USB
// iIndex=2 => 3. ...

// CH341OpenDevice: Verbindung mit CH341A herstellen
Function CH341OpenDevice(iIndex: cardinal): integer; Stdcall; external 'CH341DLL.DLL' ;
// CH341CloseDevice: Verbindung mit CH341A beenden
procedure CH341CloseDevice(iIndex: cardinal); Stdcall; external 'CH341DLL.DLL';
// CH341ResetDevice: CH341A zurücksetzen auf Einschaltzustand
Function CH341ResetDevice(iIndex: cardinal): boolean; Stdcall; external 'CH341DLL.DLL';
// CH341SetExclusive: Weiteres öffnen verhindern.
// iExclusive: 0=shared ; mehrfaches öffen möglich
//             1=exclusive; kein weiteres Öffnen möglich
Function CH341SetExclusive(iIndex:cardinal; iExclusive:cardinal ):boolean;Stdcall; external
'CH341DLL.DLL';

// **********************
// Informationen abfragen
// **********************
Function CH341GetVersion: cardinal; Stdcall; external 'CH341DLL.DLL';
Function CH341DriverCommand(iIndex: cardinal; ioCommand: mPWIN32_COMMAND):cardinal; Stdcall;
external 'CH341DLL.DLL';
Function CH341GetDrvVersion: cardinal; Stdcall; external 'CH341DLL.DLL';
Function CH341GetDeviceDescr(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG): boolean;
Stdcall; external 'CH341DLL.DLL';
Function CH341GetConfigDescr(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG): boolean;
Stdcall; external 'CH341DLL.DLL';
Function CH341GetDeviceName(iIndex:cardinal):PVOID;Stdcall; external 'CH341DLL.DLL';
Function CH341GetVerIC(iIndex:cardinal):cardinal;Stdcall; external 'CH341DLL.DLL';

// **********
// Interrupts
// **********
Function CH341SetIntRoutine(iIndex: cardinal; iIntRoutine: mPCH341_INT_ROUTINE): boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341ReadInter(iIndex: cardinal; iStatus:PULONG): boolean; Stdcall; external
'CH341DLL.DLL';
Function CH341AbortInter(iIndex: cardinal): boolean; Stdcall; external 'CH341DLL.DLL';

// Buffer (USB?) RS-232?
Function CH341SetBufUpload(iIndex:cardinal; iEnableOrClear:cardinal ):boolean;Stdcall; external
'CH341DLL.DLL';
Function CH341QueryBufUpload(iIndex:cardinal):integer;Stdcall; external 'CH341DLL.DLL';
Function CH341SetBufDownload(iIndex:cardinal; iEnableOrClear:cardinal ):boolean;Stdcall; external
'CH341DLL.DLL';
Function CH341QueryBufDownload(iIndex:cardinal ):integer;Stdcall; external 'CH341DLL.DLL';

// (USB?) RS-232?
Function CH341ResetInter(iIndex:cardinal ):boolean;Stdcall; external 'CH341DLL.DLL';
Function CH341ResetRead(iIndex:cardinal ):boolean;Stdcall; external 'CH341DLL.DLL';
Function CH341ResetWrite(iIndex:cardinal ):boolean;Stdcall; external 'CH341DLL.DLL';

// USB ? RS-232?
//Delay;  iDelay: Unit 1ms
Function CH341SetDelaymS(iIndex:cardinal; iDelay:cardinal ):boolean;Stdcall; external
'CH341DLL.DLL';
//Timeout;  iWriteTimeout: Unit 1ms
```

```
Function CH341SetTimeout(iIndex:cardinal; iWriteTimeout:cardinal; iReadTimeout:cardinal
):boolean;Stdcall; external 'CH341DLL.DLL';
Function CH341ReadData(iIndex:cardinal;      oBuffer:PVOID; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341WriteData(iIndex:cardinal; iBuffer:PVOID;     ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341FlushBuffer(iIndex:cardinal):boolean;Stdcall; external 'CH341DLL.DLL';
Function CH341WriteRead(iIndex:cardinal; iWriteLength:cardinal;    iWriteBuffer:PVOID;
iReadStep:cardinal;   iReadTimes:cardinal; oReadLength:PULONG; oReadBuffer:PVOID
):boolean;Stdcall; external 'CH341DLL.DLL' ;

// ***********************************
// Parallele Datenübertragung (EPP/MEM)
// ***********************************
// iMode: 0 = EPP/EPP V1.7 ; 1 = EPP V1.9, 2 = MEM; 256 = keep current
Function CH341InitParallel(iIndex: cardinal; iMode :cardinal): boolean; Stdcall; external
'CH341DLL.DLL';
// iMode: 0 = EPP/EPP V1.7 ; 1 = EPP V1.9, 2 = MEM
Function CH341SetParaMode(iIndex: cardinal; iMode: cardinal): boolean; Stdcall; external
'CH341DLL.DLL';
// Read & Write (MEM und EPP)
Function CH341ReadData0(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'CH341DLL.DLL';
Function CH341ReadData1(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'CH341DLL.DLL';
Function CH341AbortRead(iIndex: cardinal): boolean; Stdcall; external 'CH341DLL.DLL';
Function CH341WriteData0(iIndex:cardinal; iBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'CH341DLL.DLL';
Function CH341WriteData1(iIndex:cardinal; iBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'CH341DLL.DLL';
Function CH341AbortWrite(iIndex:cardinal): boolean; Stdcall; external 'CH341DLL.DLL';

// EPP Read/Write
Function CH341EppReadData(iIndex:cardinal; oBuffer:PVOID; ioLength:PULONG ):boolean; Stdcall;
external 'CH341DLL.DLL';
Function CH341EppReadAddr(iIndex:cardinal; oBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341EppWriteData(iIndex:cardinal; iBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341EppWriteAddr(iIndex:cardinal; iBuffer:PVOID; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341EppSetAddr(iIndex:cardinal; iAddr:byte ):boolean;Stdcall; external 'CH341DLL.DLL';

// MEM Read/Write
Function CH341MemReadAddr0(iIndex:cardinal; oBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341MemReadAddr1(iIndex:cardinal; oBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
function CH341MemWriteAddr0(iIndex:cardinal; iBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';
Function CH341MemWriteAddr1(iIndex:cardinal; iBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'CH341DLL.DLL';

// ****************************************
// Serielle Datenübertragung (I2C, RS232, SPI)
// ****************************************

// CH341SetStream() konfiguriert I2C und SPI
// Bit 1-0: I2C speed 00= low speed /20KHz
//                    01= standard /100KHz
//                    10= fast /400KHz
//                    11= high speed /750KHz

// Bit 2: SPI - Modus
//                      0= Standard SPI (D3=Takt, D5=ser. Ausgang, D7 ser. Eingang
//                      1= SPI mit doppelten Datenleitungen (D3=Takt, D5,D4=ser. Ausgänge, D7,D6
ser. Eingänge)

// Bit 7: SPI           0= LSB first
//                      1= MSB first

// andere Bits          0 (reserviert)
Function CH341SetStream(iIndex:cardinal; iMode:cardinal):boolean;Stdcall; external
'CH341DLL.DLL';

// ********************************
// SPI - Serial Peripheral Interface
// ********************************
//
```

```
// CH341A ist immer MASTER
//
// CH341BitStreamSPI()
// Serieller Datenaustausch mit (nicht SPI-konformen) seriellen IC´s (z.B. LTC1257)
// Gleichzeitiges Lesen und Schreiben über die Bytes im ioBuffer-Byte-Array:

// Bit7: D7 <- Eingang (Lesen von Peripherie-IC)
// Bit6: D6 <- Eingang (Lesen von Peripherie-IC)
// Bit5: D5 -> Ausgang (zum Peripherie-IC)
// Bit4: D4 -> Ausgang (zum Peripherie-IC)
// Bit3: D3 -> Ausgang (SCLK; serieller Takt wird von CH341A automatisch erzeugt);
// Bit2: D2 -> Ausgang (zum Peripherie-IC)
// Bit1: D1 -> Ausgang (zum Peripherie-IC)
// Bit0: D0 -> Ausgang (zum Peripherie-IC)

// Vor Aufruf der Funktion das gewünschte Signalmuster (Pattern) in die Bits D0,D1,D2,D4,D5
einsetzen
// Bei Aufruf der Funktion werden die Daten SERIELL ausgegeben (D0..D5) UND eingelesen (D6,D7)
// Nach dem Aufruf enthalten die Bits D6 und D7 die gelesenen Daten.
// D3 stellt automatisch ein Taksignal bereit. (Positiver Takimpuls für jedes Byte im IO-Buffer)
// Vor dem Aufruf aufruf müssen die Ausgänge aktiviert werden mit Set_D5_D0(iIndex, $3F)
//D0...D5 als Ausgang
// iLength ist die Anzahl der Byte die gelesen/geschrieben werden.
Function CH341BitStreamSPI(iIndex:cardinal; iLength:cardinal; ioBuffer:PVOID):boolean;Stdcall;
external 'CH341DLL.DLL';


// CH341StreamSPI4()
// Standard-SPI (Mode 0) mit zwei unidirektionalen Datenleitungen, Clock und Chip-Select.
// z.B. Atmel, etc.
// D7 <- Eingang (Lesen von Peripherie-IC; MISO)
// D5 -> Ausgang (Schreiben zum Peripherie-IC; MOSI)
// D3 -> Ausgang (SCLK; serieller Takt wird von CH341A automatisch erzeugt);
// D2 -> Ausgang (zum Peripherie-IC; Chip Select 2)
// D1 -> Ausgang (zum Peripherie-IC; Chip Select 1)
// D0 -> Ausgang (zum Peripherie-IC; Chip Select 0)
Function CH341StreamSPI4(iIndex:cardinal; iChipSelect:cardinal; iLength:cardinal; ioBuffer:PVOID
):boolean;Stdcall; external 'CH341DLL.DLL';

// CH341StreamSPI3()
// SPI mit nur EINER bi-direktionalen(!) Datenleitung, Clock und Chip-Select
// z.B. bei Maxim-Chips (DS1626); (Schreiben mit steigendem Takt; Lesen mit fallendem Takt)
Function CH341StreamSPI3(iIndex:cardinal; iChipSelect:cardinal;    iLength:cardinal;
ioBuffer:PVOID):boolean;Stdcall; external 'CH341DLL.DLL';

// CH341StreamSPI5()
// SPI mit 4 Datenleitungen (2 Paare unidirektional); Beispiel nicht bekannt
Function CH341StreamSPI5(iIndex:cardinal; iChipSelect:cardinal; iLength:cardinal; ioBuffer:PVOID;
ioBuffer2:PVOID ):boolean;Stdcall; external 'CH341DLL.DLL';

// ***********
// *** I2C ***
// ***********
//
// CH341A ist immer MASTER
//
// CH341StreamI2C(): Universelles Lesen und/oder Schreiben über I2C
// Nur Lesen: iWriteLength = 0;
// Nur Schreiben: iReadLength = 0;
Function CH341StreamI2C(iIndex:cardinal; iWriteLength:cardinal; iWriteBuffer:pvoid;
iReadLength:cardinal; oReadBuffer:pvoid ):boolean;stdcall; external 'CH341DLL.DLL';

// Lesen und Schreiben von IC mit interner Speicheradresse
// iDevice: 0..127 = I2C-Geräteadresse (device address)
// iAddr: Speicheradresse; Dies ist NICHT die I2C-Geräteadresse!
Function CH341ReadI2C(iIndex:cardinal; iDevice: byte; iAddr: byte; oByte: pbyte ):boolean;
Stdcall; external 'CH341DLL.DLL';
Function CH341WriteI2C(iIndex:cardinal; iDevice :byte; iAddr: byte; iByte: byte ):boolean;
Stdcall; external 'CH341DLL.DLL';
// I2C-EEPROM lesen und schreiben
type EEPROM_TYPE
=(ID_24C01,ID_24C02,ID_24C04,ID_24C08,ID_24C16,ID_24C32,ID_24C64,ID_24C128,ID_24C256,ID_24C512,ID
_24C1024,ID_24C2048,ID_24C4096);
Function CH341ReadEEPROM(iIndex:cardinal; iEepromID:EEPROM_TYPE; iAddr:cardinal;
iLength:cardinal; oBuffer:Pbytearray ):boolean;stdcall; external 'CH341DLL.DLL';
Function CH341WriteEEPROM(iIndex:cardinal; iEepromID:EEPROM_TYPE; iAddr:cardinal;
iLength:cardinal; iBuffer:pbytearray ):boolean;stdcall; external 'CH341DLL.DLL';

// ************************************
```

```
// Direktes Setzen der I/O Pins des CH341A
// ***************************************

// iEnable:       Sperrt bzw. erlaubt die Umschaltung der Datenrichtung
//                Bit gesetzt = Änderung zugelassen
//                Bit gelöscht = Änderungen werden ignoriert
// iSetDirOut:    Umschaltung der Datenrichtung: Eingang oder Ausgang
//                Bit gesetzt  = Pin arbeitet als Ausgang (Vorsicht!)
//                Bit gelöscht = Pin arbeitet als ist Eingang
// iSetDataOut:   Ausgänge schalten
//                Bit gesetzt  = Ausgang HIGH schalten
//                Bit gelöscht = Ausgang LOW schalten

// Zuordnung der Bit zu den Pins des CH341A:

// Bidirektionale Pins (programmierbarer Eingang oder Ausgang):
// Bit 0-7  = D0-D7          pin 15-22
// Bit 8    = ERR#           pin 5
// Bit 9    = PEMP           pin 6
// Bit 10   = ACK            pin 7
// Bit 11   = SLCT           pin 8
// Bit 12   = unused         -
// Bit 13   = BUSY/WAIT#     pin 27
// Bit 14   = AUTOFD#/DATAS# pin 4
// Bit 15   = SLCTIN#/ADDRS# pin 3

// quasi-bidirektional (Kombination aus Eingang und Open-Collector-Ausgang) :
// Bit 23   = SDA            pin 23 (bei Funktionen GetInput/GetStatus)
// Bit 16   = SDA            pin 23 (bei Funktion SetOutput)

// Uni-direktionale Pins (immer Ausgang):
// Bit 16   = RESET#         pin 26
// Bit 17   = WRITE#         pin 25
// Bit 18   = SCL pin        pin 24

// Anmerkung: Der Aufruf anderer API-Funktionen (z.B. RS232, SPI, ...
// kann den Zustand der Leitungen ändern! Das Aufeinandertreffen von zwei Ausgängen
// kann zur Beschädigung des Chips führen. Rufen Sie daher stets nur Funktionen auf
// deren Funktionsweise Ihnen bekannt ist und die zur externen Beschaltung des Chips passen.

//Ausgänge setzen
Function CH341SetOutput(iIndex:cardinal; iEnable:cardinal;
iSetDirOut:cardinal;iSetDataOut:cardinal ):boolean;Stdcall; external 'CH341DLL.DLL';
Function CH341Set_D5_D0(iIndex:cardinal; iSetDirOut:cardinal; iSetDataOut:cardinal
):boolean;Stdcall; external 'CH341DLL.DLL';
// Eingänge lesen
Function CH341GetInput(iIndex:cardinal; iStatus:PULONG ):boolean;Stdcall; external
'CH341DLL.DLL';
Function CH341GetStatus(iIndex:cardinal; iStatus: PULONG ): boolean; Stdcall; external
'CH341DLL.DLL';

// ******
// RS-232
// ******

// iParityMode:
// 0 = NONE
// 1 = ODD
// 2 = EVEN
// 3 = MARK
// 4 = SPACE
// iBaudRate:
// 50, 75, 100, 110, 134.5, 150, 300, 600, 900, 1200, 1800, 2400, 3600, 4800,
// 9600, 14400, 19200, 28800, 33600, 38400, 56000, 57600, 76800, 115200, 128000,
// 153600, 230400, 460800, 921600, 1500000, 2000000
Function CH341SetupSerial(iIndex:cardinal; iParityMode:cardinal; iBaudRate:cardinal
):boolean;Stdcall; external 'CH341DLL.DLL';
```