

# **Laborprojekt**

## **„USB-Programmer“**

(für Atmel AT89C2051/4051 Controller)

Andreas Schibilla (ii4900)  
FH-Wedel, © 2005

# 1. Inhaltsverzeichnis

1. Inhaltsverzeichnis .....	2
2. Einleitung .....	4
3. Benutzerhandbuch .....	4
3.1 Systemvoraussetzungen .....	4
3.2 Anschluss der Hardware.....	4
3.3 Treiber-Installation.....	5
3.4 Software-Installation .....	7
3.5 LED Anzeigen auf dem Gerät .....	7
3.6 Bedienung der Windows-Anwendung (GUI) .....	7
3.6.1 Laden einer Intel-Hex- oder Binärdatei .....	8
3.6.2 Ermitteln des eingelegten Chiptyps.....	8
3.6.3 Löschen und Programmieren des Flash-Speichers.....	8
3.6.4 Setzen der Lockbits .....	9
3.7 Bedienung der Windows-Konsolenanwendung .....	9
3.8 Fehlermeldungen .....	9
4. Hardwarehandbuch.....	10
4.1 Die USB-Schnittstelle .....	10
4.1.1 Technische Daten .....	10
4.1.2 Gehäuseformen und Stecker.....	11
4.2 Schaltplan .....	11
4.2.1 Schaltungsplan .....	11
4.2.2 Stückliste .....	12
4.2.3 Platinenlayout.....	13
4.3 Der Steuercontroller Atmel AT89C5131 .....	14
4.3.1 Beschaltung und Pinbelegung .....	14
4.3.2 Bereitstellung der Versorgungsspannung aus dem USB.....	15
4.3.3 Die Statusanzeige über LEDs.....	16
4.4 Erzeugen von +12V aus den +5V des USB .....	16
4.5 Umschalten der Programmierspannung (0V, +5V, +12V) .....	16
5. Programmierhandbuch .....	18
5.1 Entwicklungskonfiguration .....	18
5.2 Einspielen der AT89C5131-Firmware mittels „ausprog.exe“ .....	19
5.3 Die Kommunikation zwischen PC und Gerät .....	20
5.3.1 Allgemeiner Ablauf der Kommunikation .....	20
5.3.2 Aufbau der verschickten Pakete .....	21
5.4 Das Intel-HEX-Format .....	23
5.5 Die Firmware des Steuercontrollers AT89C5131 .....	24
5.5.1 Projektgliederung und der Kompilationsvorgang .....	24
5.5.2 Problemanalyse und grundsätzlicher Programmaufbau .....	25
5.5.3 Der USB-Enumerationsprozess.....	26
5.5.3.1 Der Ablauf der Enumeration .....	26
5.5.3.2 Die wichtigsten Deskriptoren.....	27
5.5.3.3 Die wichtigsten Befehlsanforderungen (USB-Standard-Requests) .....	27
5.5.4 Empfang und Auswertung der USB-BULK-Pakete.....	28
5.5.5 Signaturbytes vom AT89C2051/4051 auslesen .....	29
5.5.6 Löschen des Flash-Speichers vom AT89C2051/4051 .....	30
5.5.7 Auslesen und Schreiben des Flash-Speichers vom AT89C2051/4051 .....	30
5.5.8 Setzen der Lockbits vom AT89C2051/4051 .....	31
5.6 Das Windows-GUI-Programm.....	31

5.6.1 Projektübersicht und Kompilierung .....	32
5.6.2 Erstellung und Behandlung der Programmoberfläche .....	32
5.6.3 Laden und Verarbeiten neuer Hex- und Binärdateien.....	33
5.6.4 Reaktion auf Benutzereingaben, Ausführung in Threads .....	34
5.6.5 USB-Kommunikation – Senden von Befehlen .....	35
5.7 Das Windows-Kommandozeilenprogramm.....	36
5.7.1 Projektübersicht und Kompilierung .....	36
5.7.2 Die Kommandozeile.....	36
5.8 Der eingesetzte USB-Treiber .....	36
6. Anmerkungen und Probleme.....	37
6.1 Alternative Schaltungs- und Softwarevarianten.....	37
6.2 Einige Anmerkungen zu anfänglichen Problemen.....	38
6.3 Debugging-Möglichkeiten .....	39
7. Anhang .....	41
7.1 Weiterführende Literatur.....	41
7.2 Hilfreiche Internetlinks .....	42
7.3 Inhalt der CD-ROM .....	42

## 2. Einleitung

Ziel dieses Projekts ist das Erstellen eines USB-Programmers für Atmels AT89C2051/4051 Mikrocontroller. Das Gerät soll den Flash-Speicher eines eingelegten Chips beschreiben und verifizieren können, um so den fehlerfreien Upload neuer Programme zu ermöglichen.

Zum Projekt gehören die folgenden drei Teilbereiche:

- **Hardware-Entwurf** einer Schaltung, die den Programmcode über die USB-Schnittstelle vom PC empfangen kann und mit diesen Daten den eigentlichen Flashvorgang im eingelegten Chip durchführt. Die Stromversorgung erfolgt dabei direkt über die USB-Leitungen.
- Erstellen einer **Windows-Anwendung** mit zugehöriger Treibereinbindung, mit deren Hilfe der Anwender Programmcode in Form von Intel-Hex-Dateien oder Binärdateien laden und über den USB-Bus zum Gerät übertragen kann. Ebenso soll der Anwender eine Rückmeldung über den Status des Vorgangs und eventuell auftretende Fehler bekommen.
- Mikrocontroller-Programmierung der **Firmware** für das Gerät um die gewünschte Funktionalität des Programmers zu ermöglichen (Behandlung der USB-Schnittstelle, Auswertung der Befehle, Einleiten und Durchführen des Flashvorgangs)

## 3. Benutzerhandbuch

### 3.1 Systemvoraussetzungen

Für den erfolgreichen Einsatz des USB-Programmers werden folgende Hard- und Softwarekomponenten vorausgesetzt:

Hardware
<ul style="list-style-type: none"><li>• IBM-kompatibler PC mit Windows XP (NT), - mindestens eine freie USB Schnittstelle</li><li>• USB-Kabel zum Anschluss des Geräts</li><li>• Das USB-Programmer-Gerät mit eingelegtem AT89C2051/4051 Controllerchip (Stromversorgung erfolgt aus dem USB, kein Netzteil erforderlich!)</li></ul>



Software
<ul style="list-style-type: none"><li>• Betriebssystem: Windows XP (NT) mit Administratorrechten zum Installieren des USB-Treibers,</li><li>• USB-Treiber für die Kommunikation mit dem Gerät</li><li>• Windows Anwendung „usb_programmer.exe“ zur Übertragung des Programmcodes</li></ul>



### 3.2 Anschluss der Hardware

Der Programmer wird mit einem USB-Kabel an einem freien USB-Port des PCs angeschlossen. Aufgrund der „Hotplug“-Fähigkeit von USB ist das Einstecken und Rausziehen auch im laufenden Betrieb des PCs gefahrlos möglich. Sobald das Gerät angestöpselt ist (oder beim Windows-Start, wenn der PC ausgeschaltet war) erscheint ein Dialogfenster, mit der Aufforderung zur Treiberinstallation.

### 3.3 Treiber-Installation

Nachdem der PC eingeschaltet, Windows vollständig hochgefahren und alle Programme geschlossen wurden, kann das Gerät mit einem USB-Kabel angeschlossen werden. Bei der ersten Inbetriebnahme erscheint daraufhin ein kleines Infowindow am unteren Bildschirmrand (rechts abgebildet), das die neu angeschlossene Hardware meldet und zur Treiberinstallation auffordert:



Abbildung 3-1

#### Schritt 1:

Der Assistent zur Treiberinstallation wird automatisch gestartet. Die CD-ROM mit dem Treiber muss eingelegt werden. Die schnellste Möglichkeit den Treiber zu installieren bietet die Option „*Software von einer Liste oder bestimmten Quelle installieren*“.



Abbildung 3-2

#### Schritt 2:

Es kann entweder ein Laufwerk angegeben werden, in dem der Treiber gesucht wird oder das Verzeichnis wird direkt angegeben. Die entsprechende Option lautet dann „*Nicht suchen, sondern den zu installierenden Treiber selbst wählen*“.

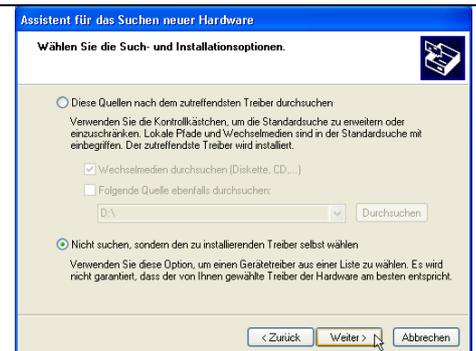


Abbildung 3-3

#### Schritt 3:

Das Verzeichnis, in dem sich der Treiber befindet, muss ausgewählt werden. Wenn das CD-ROM-Laufwerk den Laufwerksbuchstaben „D:“ besitzt, lautet der Pfad typischerweise: „*D:\Treiber*“.



Abbildung 3-4

#### Schritt 4:

Die kompatiblen Modelle für das Gerät werden aufgelistet. Da nur ein Modell zur Auswahl steht, kann über den Button „Weiter“ direkt zum nächsten Schritt gesprungen werden.



Abbildung 3-5

#### Schritt 5:

Es erscheint nun ein Fenster mit dem Warnhinweis, dass der Treiber den Windows-Logo-Test nicht bestanden hat. Diese Meldung kann ignoriert und die Installation fortgesetzt werden (Button: „Installation fortsetzen“).



Abbildung 3-6

#### Schritt 6:

Der Treiber wurde erfolgreich installiert. Der Assistent kann über den Button „Fertig stellen“ geschlossen werden!



Abbildung 3-7

Die Treiberinstallation ist nun abgeschlossen und der USB-Programmer kann verwendet werden (Installation der zugehörigen Software im nächsten Abschnitt). Beim Neustart oder nach dem erneuten Anstöpseln des Geräts am selben PC muss der Treiber nicht erneut von der CD installiert werden. Windows hat den Treiber im Systemordner zwischengespeichert und wird ihn automatisch bei Bedarf laden.

Im Gerätemanager in den Systemsteuerungen taucht das Gerät unter dem Eintrag „USB-Controller“ auf (rechte Abbildung).

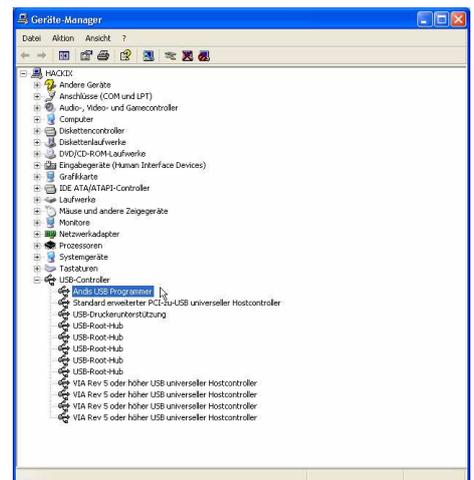


Abbildung 3-8: Der Geräte-Manager

### 3.4 Software-Installation

Neben dem Gerätetreiber sind auch zwei Windows-Anwendungen zur Steuerung des Uploads von Hex- und Binärdateien auf der beiliegenden CD-ROM enthalten („usb\_programmer.exe“).

Die eine Version besitzt eine grafische Oberfläche (Windows-GUI) und ist für die komfortable Steuerung gedacht. Die andere Version ist ein Windows-Konsolenprogramm, das nach Aufruf mit korrekten Parametern umgehend einen automatischen Upload durchführt (z.B. zum direkten Aufruf aus einem Texteditor wie UltraEdit...).

Die Programmdateien befinden sich im Verzeichnis „\Software\Windows GUI“ bzw. „\Software\Windows Konsole“ auf der CD-ROM und können mit jedem Dateimanager (z.B. Windows-Explorer) direkt auf die Festplatte kopiert werden. Ein eigenes Installationsprogramm ist nicht vorhanden!

Neben der Software sind noch weitere nützliche Tools und Informationen auf der CD-ROM, eine Übersicht steht in Kapitel „7.3 Inhalt der CD-ROM“.

### 3.5 LED Anzeigen auf dem Gerät

● Grüne LED leuchtet	Power-On:	Gerät ist betriebsbereit
● Gelbe LED leuchtet	Busy:	Gerät arbeitet
● Rote LED leuchtet	Fehler:	Befehl war nicht erfolgreich

### 3.6 Bedienung der Windows-Anwendung (GUI)

Für den USB-Programmer stehen zwei Windowsanwendungen für die Uploadsteuerung bereit. Eine grafische Version (GUI-Anwendung), die in diesem Kapitel genauer beschrieben wird, und eine Konsolenanwendung für das automatische Flashen z.B. direkt aus einem Editor heraus (→ siehe nächstes Kapitel).

Im Verzeichnis „Software\Windows GUI“ auf der CD-ROM befindet sich die Datei „usb\_programmer.exe“.

Die rechte Abbildung 3-9 zeigt die Oberfläche des Programms.

Zu Beginn muss eine Hex-Datei (oder ein binäres Speicherabbild) geladen werden. Dann können über die gezeigten Buttons verschiedene Befehle an den USB-Programmer abgesetzt werden. Wurde noch keine gültige Hex-Datei geladen, so bleiben einige Funktionen/Buttons deaktiviert und können nicht ausgeführt werden.

Im Log-Fenster (im unteren Drittel der Oberfläche) werden die durchgeführten Operationen, Statusmeldungen und evtl. auftretende Fehler angezeigt.

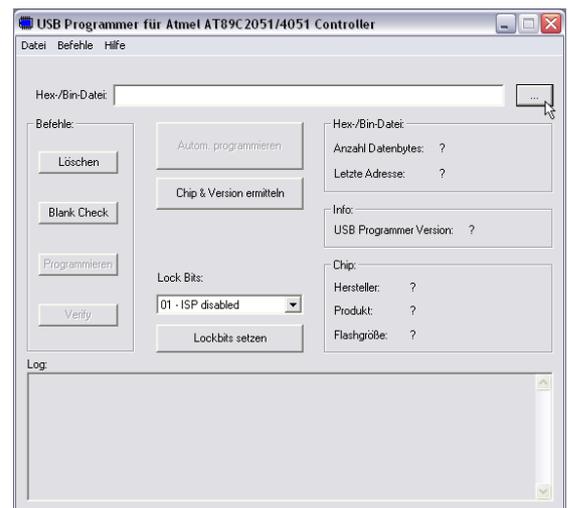


Abbildung 3-9: Das Windows GUI Programm

### 3.6.1 Laden einer Intel-Hex- oder Binärdatei

Nachdem das Windowsprogramm gestartet wurde, muss eine Hex-Datei (oder Binärdatei), die alle Daten für den Flashvorgang enthält, geladen werden. Dies geschieht entweder über den „Durchsuchen-Button“  oder den Menüaufruf „Datei → Lade Hex- oder Binärdatei“ (rechts abgebildet).



Daraufhin erscheint ein Dateiauswahldialog, mit dem die gewünschte Datei geöffnet werden kann. Im Auswahlfeld „Dateityp“ kann der Filter für die Dateierweiterung (Binär- oder Hex-Datei) angegeben werden. Nach Klick auf den „Öffnen-Button“ wird die Datei überprüft und anschließend geladen. Die Anzahl der Datenbytes sowie die letzte Speicheradresse im Chip werden angezeigt!

### 3.6.2 Ermitteln des eingelegten Chiptyps

Mit Hilfe des „Chip & Version ermitteln-Buttons“ (oder den Menüpunkt „Befehle → Chip & Version ermitteln“) kann der eingelegte Chip und die Versionsnummer des USB-Programmers ausgelesen werden. Nach Klick auf den Button erscheint einerseits eine Statusmeldung im Logfenster, andererseits steht im rechten Infofeld die USB-Programmer-Versionsnummer und Informationen zum eingelegten Chip (Hersteller, Produktbezeichnung und Flashgröße).

Diese Funktion sollte normalerweise immer vor den anderen Befehle aufgerufen werden!

### 3.6.3 Löschen und Programmieren des Flash-Speichers

Das Windows-Programm ermöglicht über die Befehl-Buttons oder den entsprechenden Einträgen im Menü „Befehle“ die folgenden Optionen:

- |   |  |
|---|--|
|  | ▪ Der komplette Flash-Speicher des eingelegten AT90C2051/4051 Controllers wird gelöscht. Evtl. gesetzte Lockbits werden ebenfalls zurückgesetzt.   |
|  | ▪ Es wird überprüft, ob der Flash-Speicher erfolgreich gelöscht wurde. Jedes Byte im Flash-Speicher des eingelegten Chips wird dabei überprüft.  |
|  | ▪ Die ausgewählte Hex- oder Binärdatei wird in den Flash-Speicher des eingelegten Controllers geschrieben. Adressen, die im Hexfile nicht angegeben sind, werden übersprungen (alter Wert bleibt erhalten)!  |
|  | ▪ Es wird überprüft, ob die aktuell ausgewählte Hex-/Binärdatei erfolgreich in den Chip übertragen wurde. Dazu werden die Hex-/Binärdaten erneut zum USB-Programmer geschickt und anschließend mit den Daten aus dem Flash-Speicher des eingelegten Controllers verglichen.  |
|  | ▪ Diese Option ermöglicht das automatische Löschen, Programmieren und Verifizieren des eingelegten AT89C2051/4051 Controllers. Die oberen vier Befehle werden dazu automatisch nacheinander durchgeführt. Zusätzlich wird hier die benötigte Zeit des Flashvorgangs gemessen und am Ende im Logfenster ausgegeben. |

### 3.6.4 Setzen der Lockbits

Die AT89C2051/4051 Controller besitzen zwei Lock Bits, mit deren Hilfe es möglich ist, das weitere Programmieren zu sperren.

Über das Auswahlmeneü „Lock Bits“ (siehe rechte Abbildung 3-10) kann die gewünschte Einstellung angegeben und durch Klick auf den Button „Lockbits setzen“ in den eingelegten Chip geschrieben werden. Die Lockbits können nur durch das komplette Löschen des Chips zurückgesetzt werden!



Abbildung 3-10

Die folgenden Tabelle aus dem Datenblatt zeigt die möglichen Bitkombinationen:

Program Lock Bits			Protection Type
	LB1	LB2	
1	U	U	No program lock features
2	P	U	Further programming of the Flash is disabled
3	P	P	Same as mode 2, also verify is disabled

### 3.7 Bedienung der Windows-Konsolenanwendung

Im Verzeichnis „\Software\Windows Konsole\“ auf der CD-ROM befindet sich die Konsolenanwendung „usb\_programmer.exe“. Mit diesem Tool ist es möglich, eine automatische Programmierung eines Chips von der Kommandozeile aus zu starten. Somit lässt sich der Flash-Vorgang z.B. in die Werkzeugleiste eines Editors wie UltraEdit oder SciTE einbinden.

Aufrufparameter:

Syntax: *usb\_programmer.exe Hexfile*

**Hexfile** kann eine Intel-Hex-Datei sein (Dateiextension: .hex oder .ihx)  
oder eine Binärdatei (andere Dateiextension)

### 3.8 Fehlermeldungen

Im Umgang mit der Windowsanwendung des USB-Programmers können folgende Fehlermeldungen auftreten:

Fehlermeldung	Ursache / Behebung:
ERR: Ungültige Hex-Datei! Fehler in Zeile x	Die Hexdatei ist nicht im gültigen Intel-Hex-Format! Überprüfen Sie die Dateiangaben und das Zeilenformat in der Zeile x.
ERR: Konnte Hex-Datei <Dateiname> nicht zum Lesen öffnen!	Überprüfen Sie die Pfadangaben, ist die Datei vorhanden und wird nicht von einer anderen Anwendung verwendet (und ist somit gesperrt)?
ERR: Ungültige Datei ausgewählt!	Ihre gewählte Binärdatei enthält kein gültiges Speicherabbild für den Chip. Vermutlich ist die Datei zu groß oder konnte nicht zum Lesen geöffnet werden.

Zu Beachten sind außerdem alle Meldungen im Logfenster! Befehlsspezifische Hinweise und Fehler werden dort angezeigt!

## 4. Hardwarehandbuch

Im folgenden Abschnitt dieser Dokumentation wird die Hardware des USB-Programmers beschrieben und die Funktionsweise erklärt. Zunächst wird die USB-Schnittstelle vorgestellt, darauf folgt dann ein Einblick in den kompletten Schaltungsplan, der schließlich in seine einzelnen Funktionsblöcke zerlegt und erläutert wird.

### 4.1 Die USB-Schnittstelle

#### 4.1.1 Technische Daten

Die USB-Schnittstelle besitzt im Vergleich zu den älteren RS-232 oder Centronics Anschlüssen viele Vorteile, die folgende Liste zeigt einige Features:

- Leichte Anwendbarkeit, automatische Konfiguration nach Anschluss, keine IRQ-Einstellungen vom Anwender notwendig
- BUS-Konzept ermöglicht das Anschließen von bis zu 127 Geräten
- Dünne Kabel und handlichere Stecker
- Hot Plugging ermöglicht das beliebige An- oder Abstecken von Peripheriegeräten, wobei es keine Rolle spielt, ob der PC eingeschaltet ist oder nicht, da das laufende Betriebssystem die angeschlossene Hardware erkennt und initialisiert
- USB-Geräte benötigen oftmals keine eigene Stromversorgung, da die USB-Schnittstelle +5V Stromversorgungs- und Masseleitungen bereitstellt (typ. 100mA, nach besonderer Anmeldung bis max. 500mA)
- USB besitzt eine kaskadierte Stern-Topologie, d.h. an einen HUB in der Mitte können sowohl Geräte als auch weitere HUBs angeschlossen werden
- Flexibilität, Zuverlässigkeit durch Datenfehlererkennung und rel. niedriger Stromverbrauch  
Aber: Komplexes Protokoll erfordert höhere Anforderungen an die Entwickler

#### USB unterstützt drei Busgeschwindigkeiten:

- Low-Speed (USB 1.0)      1,5 Mbit/s ( $\approx$  800 Byte/s)
- Full-Speed (USB 1.0)    12 Mbit/s ( $\approx$  1,2 MByte/s)
- High-Speed (USB 2.0)    480 Mbit/s ( $\approx$  53 MByte/s)

#### Pinbelegung:

Die USB-Schnittstelle umfasst 4 Leitungen mit folgender Pinbelegung:

- |   |        |           |
|---|--------|-----------|
| 1 | +5V    | (rot)     |
| 2 | Data - | (weiss)   |
| 3 | Data + | (grün)    |
| 4 | Masse  | (schwarz) |

#### Spannungspegel:

- Die Versorgungsspannung am USB beträgt typ. 4,2V (bei starker Belastung) bis 5,25V
- Auf den beiden Leitungen D+ und D- werden Differenzsignale mit Spannungspegeln von 0V/3,3V übertragen.

## 4.1.2 Gehäuseformen und Stecker

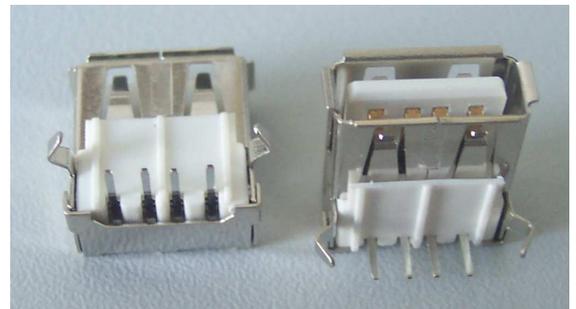
Es gibt zwei verschiedene USB-Steckertypen: Typ A und Typ B. Am PC sind meistens Buchsen vom Typ A zu finden, die Peripheriegeräte haben hingegen häufig eine USB-Buchse vom Typ B (wie auch dieser USB-Programmer). Die Verbindung erfolgt mit Hilfe eines „normalen“ USB-Kabels (vom Typ A-B).



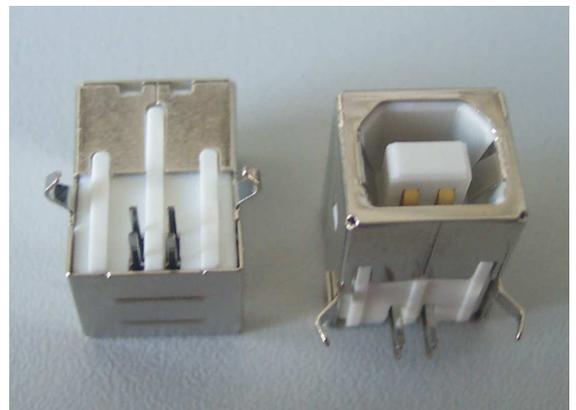
Stecker Typ-A



Stecker Typ-B



Buchse Typ-A



Buchse Typ-B

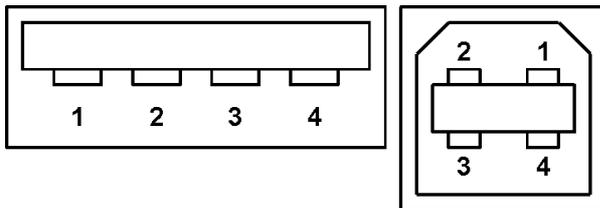


Abb. 4-1: USB-Typen von Steckern und Buchsen

## 4.2 Schaltplan

### 4.2.1 Schaltungsplan

Die Abbildung 4-2 auf der nächsten Seite zeigt den vollständigen Schaltungsplan des USB-Programmers. Der in der rechten Hälfte eingezeichnete Controller AT89C4051 gehört natürlich nicht mit zur eigentlichen Schaltung des Geräts. An dieser Stelle befindet sich stattdessen ein Text-tool-Sockel, in den ein zu „flashender“ Chip eingelegt werden kann, so dass sich die abgebildeten Verknüpfungen ergeben.

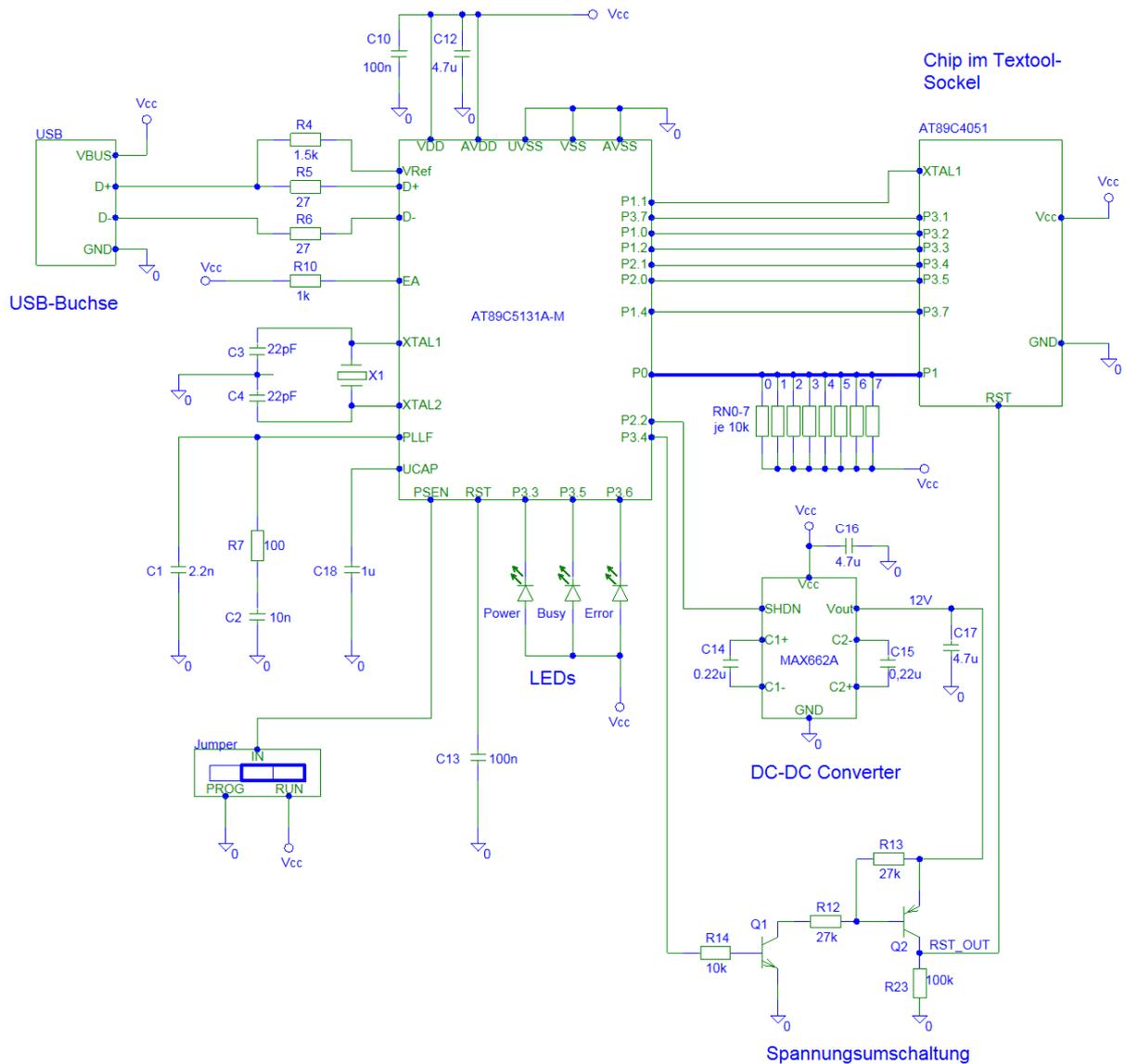


Abb. 4-2: Schaltungsplan

## 4.2.2 Stückliste

### Halbleiter:

1 x µController Atmel AT89C5131A-M	{ D2 }
1 x µController Atmel AT89C2051/4051	{ D1 }
1 x DC-DC Converter MAX662A	{ D4 }
1 x NPN-Transistor z.B: BC107A oder BC547	{ V1 }
1 x PNP-Transistor z.B. BC177A oder BC559	{ V2 }
1 x LED 3mm, grün	{ H3 }
1 x LED 3mm, gelb	{ H2 }
1 x LED 3mm, rot	{ H1 }

### Widerstände:

2 x 27Ω	{ R8, R9 }
1 x 100Ω	{ R7 }
1 x 1kΩ	{ R6 }
1 x 1,5kΩ	{ R10 }
1 x 10kΩ	{ R3 }
2 x 27kΩ	{ R1, R2 }
1 x 10kΩ-Widerstandsnetzwerk (8-1)	{ RN1 }

### Kondensatoren:

2 x 22pF	{ C3, C4 }
1 x 2,2nF	{ C13 }
1 x 10nF	{ C14 }
2 x 100nF	{ C5, C7 }
2 x 0,22µF	{ C11, C12 }
1 x 1µF (Elko)	{ C15 }
3 x 4,7µF (Elko)	{ C6, C9, C10 }

### Sonstiges:

1 x Quarz, 16MHz	{ G1 }
1 x Steckerleiste (3 Pins) + Jumper (2 Pins)	{ X1 }
1 x USB-Buchse, Typ-B (z.B. ASSMANN)	{ X2 }
1 x IC-Sockel (8 Pins) für MAX662A	{ D4 }
1 x IC-Sockel (PLCC-52) für AT89C5131	{ D2 }
1 x Texttool-Sockel, 20-polig + IC-Sockel	{ D1 }
4 x Schraube BO-M3.0-S	{ SC1-4 }
1 x Platine und pass. Gehäuse	

### 4.2.3 Platinenlayout

Die beiden folgende Abbildung zeigen das Platinendesign. Die Platine ist so aufgebaut, dass sie problemlos in ein 10,2x6,1cm Gehäuse passt (z.B. bei Reichelt zu beziehen).

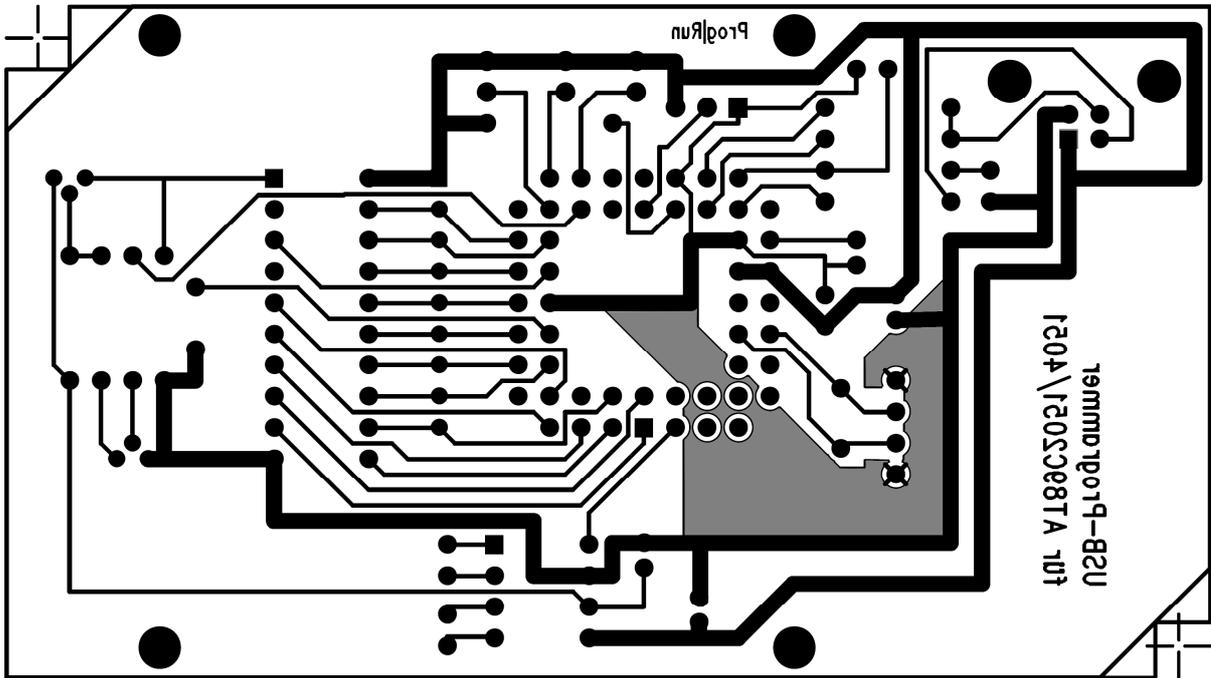


Abbildung 4-3: Unterseite der Platine (Bottom Layer)

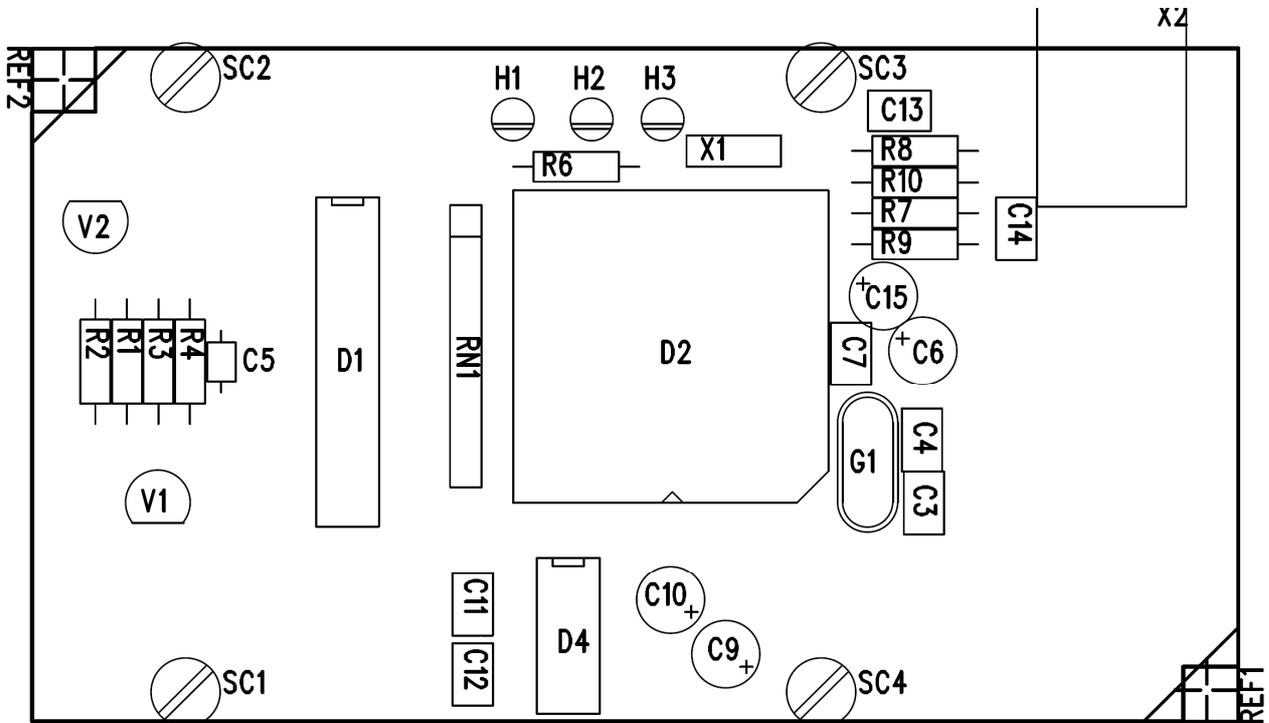


Abbildung 4-4: Bestückungsseite der Platine

### 4.3 Der Steuercontroller Atmel AT89C5131

Der Atmel Controller AT89C5131 stellt das Herzstück des USB-Programmers dar. Er enthält ein Mikroprogramm, das die gesamte USB-Kommunikation und den Flash-Vorgang des eingelegeten Chips steuert.

#### 4.3.1 Beschaltung und Pinbelegung

Um den Atmel AT89C5131 Controller mit der USB-Schnittstelle betreiben zu können, ist die in Abb. 4-55 gezeigte, typische Beschaltung notwendig:

- **Power Supply:** Die Stützkondensatoren können bei kurzzeitigen Spannungseinbrüchen zusätzlichen Strom für den Controller bereitstellen.
- **USB Full-Speed-Connection:** Die beiden Signalleitungen des USB werden jeweils über einen  $27\Omega$ -Widerstand angeschlossen. Damit der USB-HUB im PC erkennen kann, ob das neu angeschlossene Gerät im Lowspeed- oder Fullspeed-Betrieb arbeitet, muss eine der beiden Signalleitungen über einen  $1,5k\Omega$  Widerstand mit  $+3,3V$  verbunden werden. Da dieser USB-Programmer mit Fullspeed arbeitet, wird D+ hochgezogen (bei einem Lowspeed-Gerät würde D- hochgezogen).
- **Clock Oscillator:** Die Taktfrequenz des Controllers wird mit Hilfe eines 16MHz Quarz über die X1- und X2-Pins erzeugt.
- **PLL Low Pass Filter:** Die für den USB benötigte Taktfrequenz von 48MHz wird intern mit Hilfe einer Phase-Locked-Loop aus der Quarzfrequenz erzeugt, dessen Tiefpass über das am PLLF-Pin angeschlossene RC-Netzwerk eingestellt wird.
- **Program Mode:** An dem PSEN-Pin ist ein Jumper (oder alternativ ein Schalter) angeschlossen, der den Eingang entweder mit Low (Masse) oder mit High ( $V_{dd}$ ) verbindet.
  - *PSEN = Low:* Der Controller startet nach einem Resetimpuls den internen Bootloader und ermöglicht somit seine Neuprogrammierung (Upload in Flash/EEPROM-Speicher).
  - *PSEN = High:* Der Controller startet nach einem Resetimpuls das eingespielte Programm aus seinem Flashspeicher.
- **Power-on Reset:** Der Reset-Pin des Controllers besitzt einen internen Pull-Up-Widerstand. Mit Hilfe des Kondensators, der vom Reset-Pin auf Masse geschaltet ist, wird somit der Resetimpuls beim Einschalten erzeugt.
- **I/O-Ports:** In der rechten Hälfte der Abbildung sind die für die Schaltung relevanten Ports eingezeichnet.

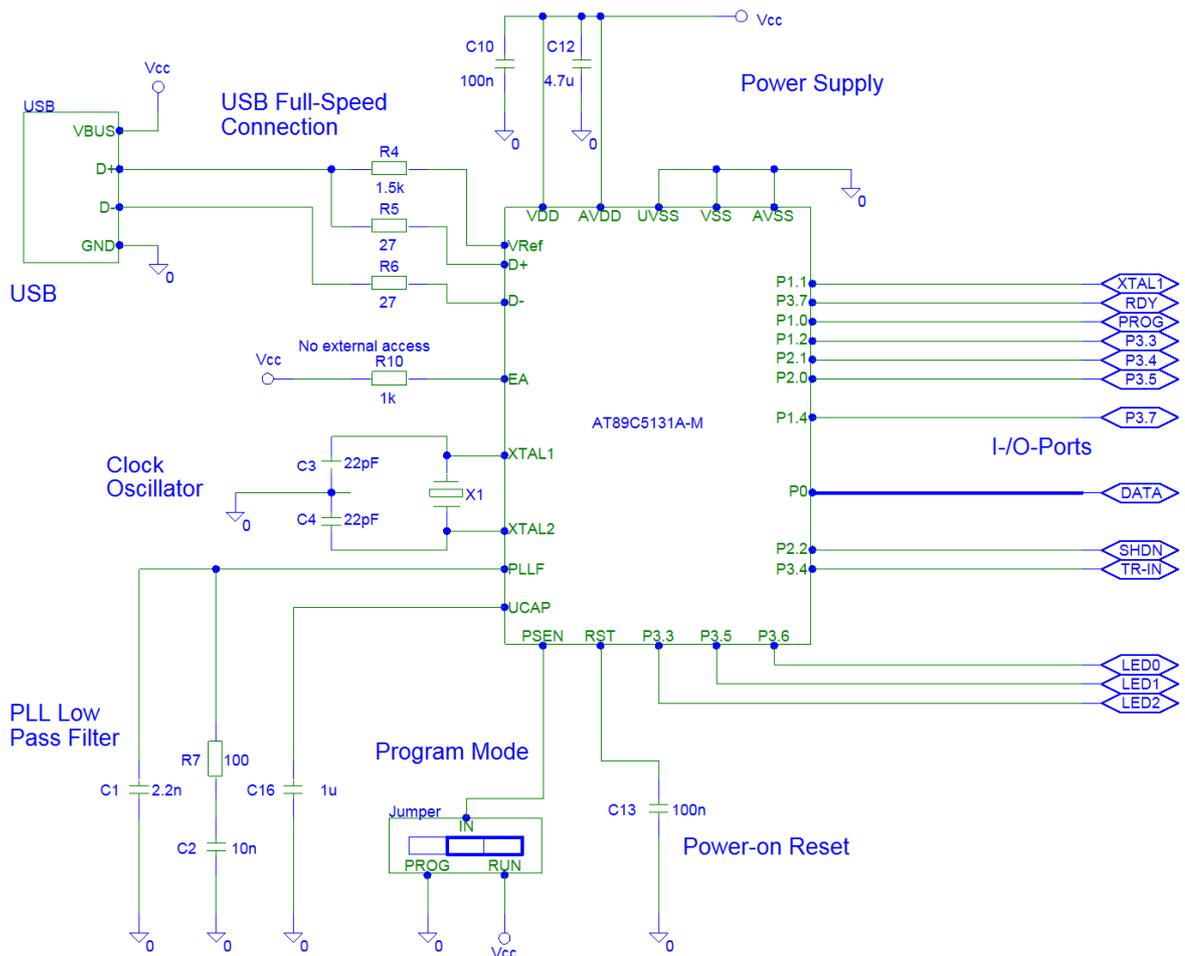


Abb. 4-5: Typische Beschaltung des AT89C5131 (und Aufführung der I/O-Ports)

### 4.3.2 Bereitstellung der Versorgungsspannung aus dem USB

Der Controller AT89C5131A-M arbeitet mit einer Betriebsspannung im Bereich von +3,3V bis +5,5V. Der USB-Bus liefert nahezu eine Spannung von +5V und erlaubt zunächst eine maximale Belastung von 100mA. Da die Schaltung einen weitaus geringeren Strombedarf hat, ist die direkte Speisung aus dem USB-Bus problemlos möglich, so dass kein weiteres Netzteil benötigt wird. Kleine Spannungseinbrüche werden zudem durch mehrere „Stützkondensatoren“ an den Eingangspins der Halbleiter abgefangen (→C10, C12).

### 4.3.3 Die Statusanzeige über LEDs

Der AT89C5131 Controller besitzt vier programmierbare LED Stromquellen. Jeder dieser vier Ports kann für 2mA, 4mA oder 10mA konfiguriert werden. Die LEDs werden dann direkt mit der Kathode ohne Vorwiderstand angeschlossen (Anode auf Vcc, siehe Abbildung 4-6).

Die Stromquellen werden über das Register LEDCON konfiguriert. Ist ein entspr. LED-Port auf Low gesetzt, fließt der eingestellte Strom und die LED leuchtet auf. Der folgende Quelltextauszug zeigt die Initialisierung:

```
// LED-Pins definieren
#define LED_POWER      P3_3
#define LED_BUSY       P3_5
#define LED_ERROR      P3_6

// LED's initialisieren
LEDCON = 0xFF; // alle vier LEDs auf 10mA Stromquellen
LED_POWER = 0; // Betriebslampe an
LED_BUSY = 1; // die anderen beiden LEDs aus
LED_ERROR = 1;
```

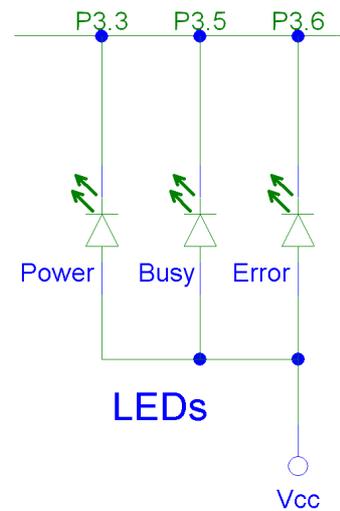


Abbildung 4-6: Die Status-LEDs

### 4.4 Erzeugen von +12V aus den +5V des USB

Während der Flashprogrammierung des Bausteins wird eine Spannung von +12V ( $\pm 0,5V$ ) benötigt.

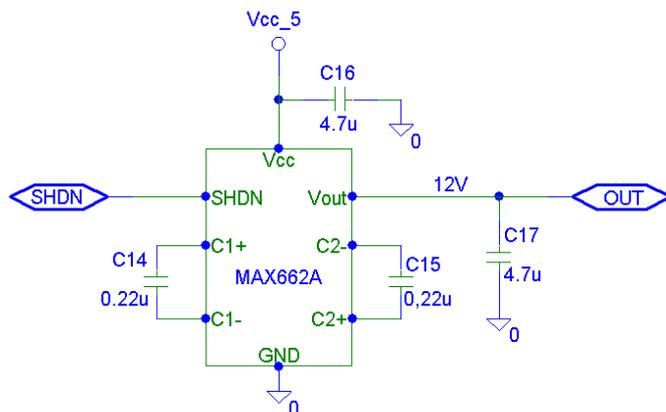


Abb. 4-7: Spannungspumpe +5V  $\rightarrow$  +12V

Da der Laststrom im Datenblatt mit  $<1mA$  angegeben ist, können die +12V mit einem DC-DC-Converter (Spannungspumpe) aus der +5V Versorgungsspannung des USB erzeugt werden.

In der Schaltung kommt dazu ein Maxim MAX662A Baustein zum Einsatz, der genau diese Aufgabe übernimmt. Abb. 4-7 zeigt die Beschaltung.

Über den Shutdown-Eingang (SHDN) kann die Ausgangsspannung auf +12V (SHDN=Low) oder +5V (SHDN=High) eingestellt werden.

### 4.5 Umschalten der Programmierspannung (0V, +5V, +12V)

Für den Flashprogrammervorgang des im Textool-Sockel eingelegten AT89C2051/4051 Controllers, werden die Spannungen 0V, +5V und +12V am Reset-Eingang benötigt. Die beiden Spannungen +5V und +12V können vom Steuercontroller AT89C5131 über den Shutdown-Eingang des MAX662A-Bausteins eingestellt werden (SHDN=Low  $\rightarrow$  +12V am Ausgang; SHDN=High  $\rightarrow$  Vcc=+5V am Ausgang). Um zusätzlich auch 0V schalten zu können, kommt folgende Transistor-schaltung zum Einsatz:

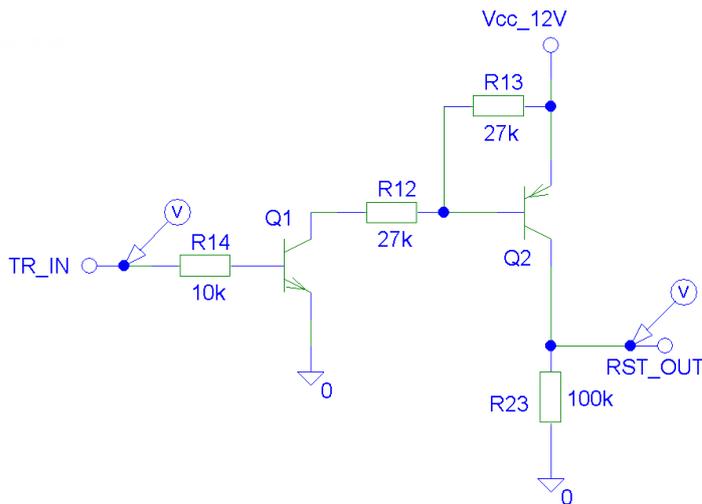


Abb. 4-8: Spannungsumschaltung 0V → 5V/12V

Erscheint ein High-Pegel ( $\approx +5V$ ) am Eingang TR\_IN, so wird der Transistor Q1 durchschalten. Da Q2 der Komplementärtyp zu Q1 ist (PNP), wird auch er leitend, so dass am Ausgang nahezu Vcc liegt (+12V oder +5V, je nach Ansteuerung des MAX662A-Bausteins).

Wird der Eingang mit Low-Pegel angesteuert, sperren beide Transistoren und der Ausgang wird über den Pulldown-Widerstand R23 auf definiertes Massepotential gezogen.

Der Widerstand R13 sorgt im gesperrten Zustand von Q1 dafür, dass an der Basis von Q2 nahezu die vollen 12V abfallen und Q2 sicher gesperrt bleibt.

Die Dimensionierung von R12-R14 ist so gewählt, dass ein gutes Schaltverhalten bei relativ geringen Strömen erzielt wird. R23 wurde deutlich höher gewählt, damit im durchgeschalteten Zustand eine möglichst geringe Spannung über  $U_{CE}$  bei Q2 abfällt ( $\rightarrow$  Spannungsteiler!).

Die untenstehenden Abbildungen demonstrieren das Ergebnis der PSpice-Simulation zur Spannungsumschaltung. Das erste Diagramm zeigt das Verhalten am Ausgang bei unterschiedlichen Eingangsspannungen. Man erkennt sehr deutlich, dass die Schaltschwelle ungefähr bei +0,5V und somit im sicheren Bereich liegt. Das zweite Diagramm zeigt das zeitliche Verhalten der Ausgangsspannung bei mehreren Schaltvorgängen am Eingang. In der dargestellten Auflösung ist keine kritische Verzögerung zu erkennen. Außerdem werden die Spannungspegel von 0V und +12V sehr gut eingehalten.

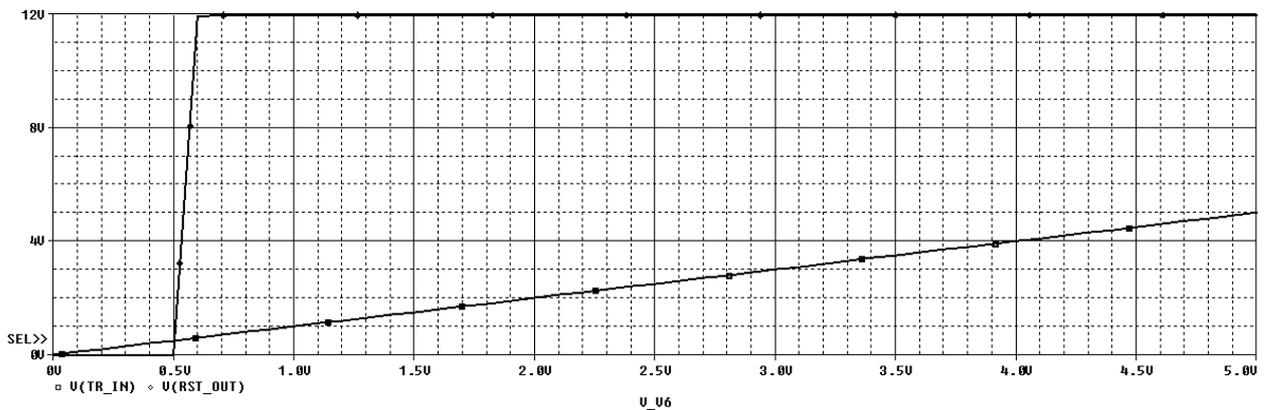


Abb. 4-9: DC Sweep; Ausgangsspannung  $V_{RST\_OUT}$  in Abh. von der Eingangsspannung  $V_{TR\_IN}$

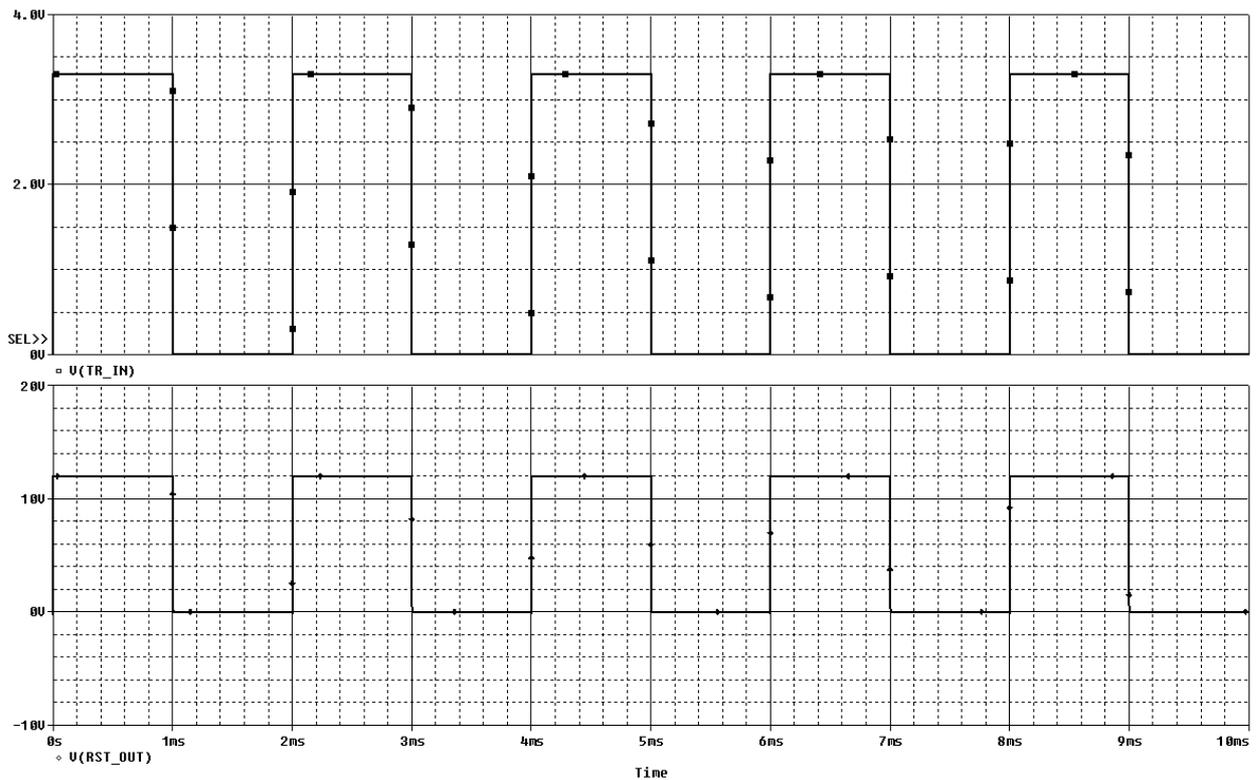


Abb. 4-10: Transientenanalyse; Umschaltverhalten in Abh. der Zeit (oben: Eingang - unten: Ausgang)

## 5. Programmierhandbuch

### 5.1 Entwicklungskonfiguration

Die Entwicklung des USB-Programmers wurde mit Hilfe folgender Konfiguration durchgeführt:

#### Hardware

- Standard-Laptop 1,4GHz Pentium-M (WinXP kompatibel)
- USB 2.0 Schnittstelle (Intel 82801DB/DBM)
- Externer, aktiver USB 4-Port HUB (Typhoon, USB2.0)  
(wird für „USBCommandVerifier“ benötigt!)

#### Software

- Betriebssystem: Windows XP (SP2)
- Programmierumgebung:
  - SDCC-Compiler, Version: May 8 2005
  - UltraEdit-32 Text Editor
  - Microsoft Visual C/C++ Version 6.0
- AT89C5131 USB-Bootloader Upload-Tool v1.2 (ausbprog.exe)
- ATMUSB-Treiber
- Diagnose-Tool: USB View und USB Browser
- Diagnose-Tool: USBCommandVerifier von [www.usb.org](http://www.usb.org)
- Hardware-Simulation: OrCAD PSpice
- Für das Platinendesign: Mentor Graphics PowerLogic bzw. PowerPCB und die entsprechenden Bohrfile-Tools an der FH-Wedel

## Anmerkung:

Der von Atmel gelieferte USB-Treiber, sowie das Atmel Upload Tool „Flip“ werden nicht benötigt und sollten auch NICHT installiert sein (da es dann zu Treiberkonflikten kommen kann)!

## 5.2 Einspielen der AT89C5131-Firmware mittels „ausbprog.exe“

Der Atmel Controller AT89C5131 unterstützt In-System-Programming (ISP), was bedeutet, dass der Chip nicht über einen externen Brenner beschrieben werden muss, sondern er kann direkt in die fertige Schaltung eingesetzt werden und dort über den USB-Bus programmiert werden. Damit das möglich ist, besitzt der Baustein einen internen Bootloader, der die USB-Kommunikation beim Starten selbständig durchführt und anschließend das Programmieren des Bausteins ermöglicht.

Ob der Chip nach einem Reset (bzw. nach dem Einschalten) mit dem Bootloader hochfährt oder das vom Anwender geschriebene Programm ausführt, wird über den PSEN-Pin und das Boot Loader Jump Bit (BLJB) gesteuert.

Der Upload eines Programms kann wie folgt erfolgen:

1. Setzen des Jumpers „X1“ auf „Prog“, damit der PSIN-Pin mit Masse verbunden wird (Chip fährt mit Bootloader hoch)
2. Verbinden des Programmiers mit der USB-Schnittstelle des PCs; die grüne Betriebslampe leuchtet nicht auf.
3. Windows fordert den Anwender ggf. auf, einen Treiber anzugeben. Die Installation kann wie in „Kap 3-3 Treiberinstallation“ vorgenommen werden
4. Nachdem Windows die Hardware erkannt hat, kann das Upload-Tool „ausbprog.exe“ (im Verzeichnis „\Tools\ AT89C5131 USB-Bootloader Upload-Tool\“ auf der CD-ROM) gestartet werden. Die folgende Abbildung zeigt die Oberfläche des Programms:

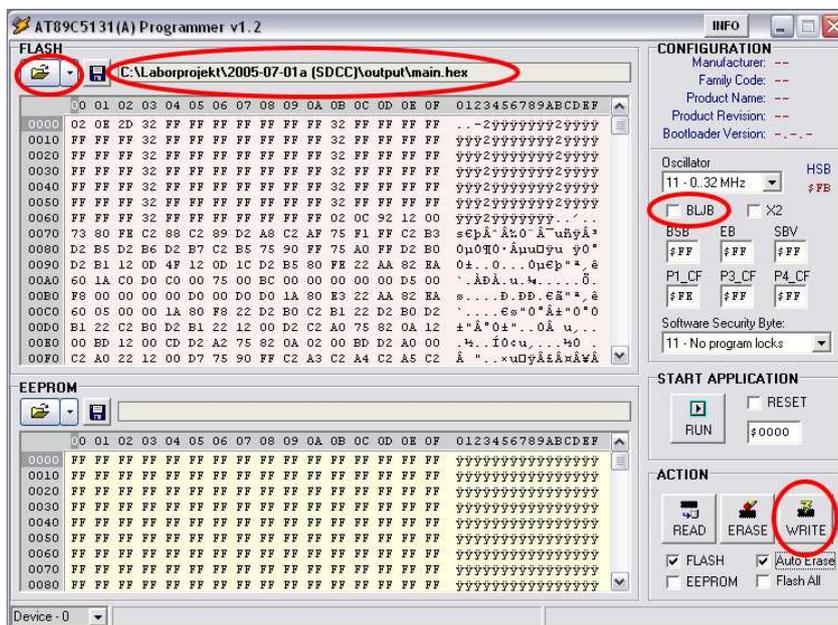


Abb. 5-1: Oberfläche von AUSBPROG.EXE

5. Nun kann eine neue Hex-Datei geladen werden, woraufhin die Bytes im Hauptfenster zu sehen sind.
6. Als nächstes wird der Uploadvorgang konfiguriert. Normalerweise können alle Standardeinstellungen beibehalten werden. Wichtig ist jedoch, dass das BLJB-Bit nicht gesetzt wird, der Haken bei der entspr. Checkbox muss also entfernt werden!

7. Schließlich kann die Hexdatei durch Klick auf den Write-Button in den Chip geschrieben werden. (→ weitere Informationen zu dem Programm stehen in der Textdatei „*readme.txt*“)
8. Nach dem Schreibvorgang muss das Gerät vom PC getrennt und somit ausgeschaltet werden. Der Jumper „*X1*“ wird wieder auf „*Run*“ zurückgesetzt. Nach dem erneuten Anschließen am PC führt der Controller nun das eingespielte Programm aus.

## 5.3 Die Kommunikation zwischen PC und Gerät

### 5.3.1 Allgemeiner Ablauf der Kommunikation

Um Daten vom PC zum USB-Gerät zu verschicken (oder zu empfangen) müssen mehrere Instanzen durchlaufen werden. Der Anwender startet die „normale“ Windowsanwendung „*usb\_programmer.exe*“, die alle Eingaben des Benutzers entgegennimmt. Es werden die zu verschickenden Daten (Hexfile) geladen und für die Übertragung vorbereitet. Für die eigentliche Übertragung wird der USB-Treiber „*atmusb.sys*“ initialisiert (CreateFile), anschließend wird mit Hilfe der Windows-API (DeviceIoControl) auf die Treiberfunktionen zugegriffen. Der Treiber wiederum kommuniziert mit dem USB-Host-Controller (HUB) des PCs, der für jeden USB-Anschluss mind. einen Port bereitstellt. Der Port ist über den USB-Bus mit mehreren logischen Verbindungen (Pipes) mit dem Controller des Geräts verbunden. Die untenstehenden Abbildung 5-2 zeigt die grundsätzlich beteiligten Instanzen bei der USB-Kommunikation.

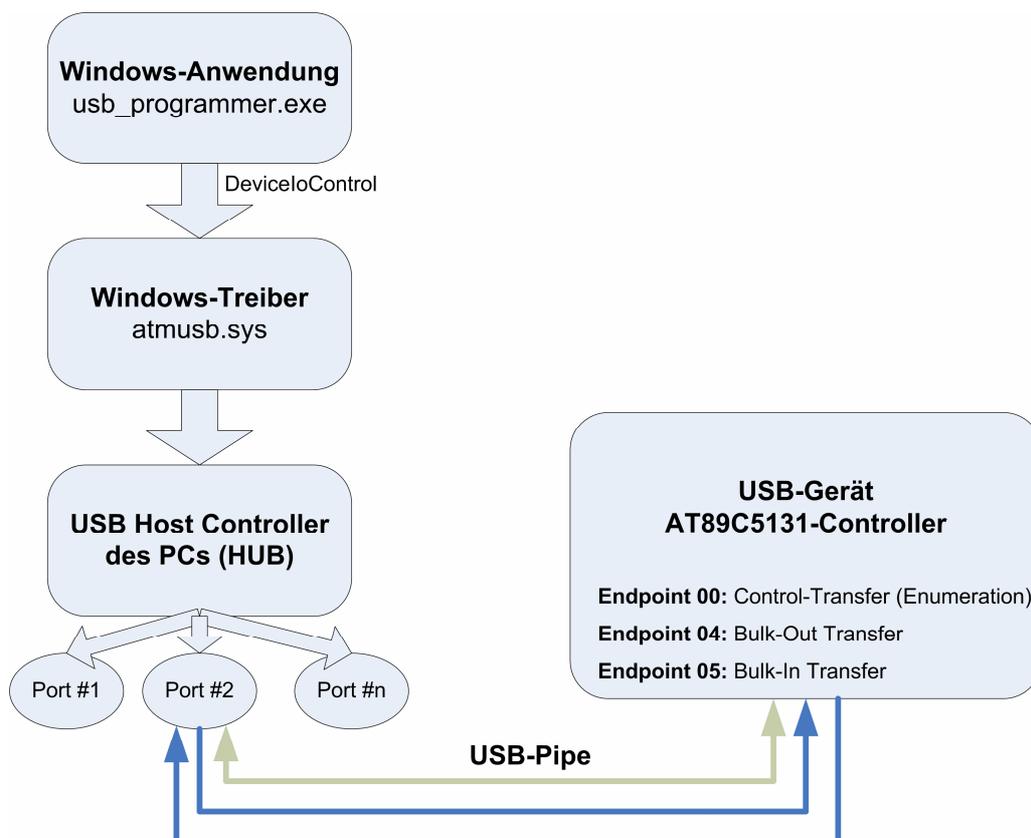


Abbildung 5-2 Ablauf der USB-Kommunikation zeigt die grundsätzlich beteiligten Instanzen bei der USB-Kommunikation

Die eigentliche Übertragung findet seriell statt. Die Serialisierung sowie das „Bit-Stuffin“ (Synchronisation etc.) wird intern vom Atmel USB-Controller übernommen. Für die Programmierung ist jedoch wichtig, dass der Controller mehrere FIFO-Speicher besitzt, an die Daten übertragen werden können. Zu jedem FIFO-Speicher gehört eine sog. Endpoint-Adresse (00-06), die angibt, wohin die Daten gelangen sollen, oder von wo sie geholt werden. Die einzelnen Endpoints werden im Programmcode des Mikrocontrollers konfiguriert.

Dieser USB-Programmer verwendet Endpoint 00 im Modus: Control-Transfer (für die Enumeration des Geräts), sowie Endpoint 04 und 05 im Bulk-Modus, den einen zum Senden, den anderen zum Empfang.

Der Datenaustausch erfolgt in kurzen Paketen von max. 64 Bytes. Wie diese Pakete aufgebaut sind, wird im nächsten Kapitel beschrieben.

### 5.3.2 Aufbau der verschickten Pakete

Die Kommunikation zwischen der PC-Software und dem USB-Programmer erfolgt in Befehlspaketen von max. 64 Bytes. Der Inhalt dieser Bytes ist in diesem Projekt wie folgt festgelegt:

Grundsätzlich entspricht das erste Byte von jedem Befehlspaket der Signaturkennung, die den Wert ‚A‘ (= 41h) haben muss. Pakete mit einem anderen Wert im ersten Byte sind ungültig! Das zweite Byte gibt an, um was für einen Befehl es sich handelt. Im Programmcode sind dafür mehrere Konstanten abgelegt. Als nächstes können befehlspezifische Datenbytes folgen.

→ Die Windowsanwendung „usb\_programmer.exe“ kann folgende Befehle an das USB-Gerät senden:

**CMD\_ERASE:** Der komplette Flash-Speicher soll gelöscht werden

<b>00h:</b>	<b>01h:</b>	
Signatur-Byte: 'A' = 41h	Befehls-Byte: CMD_ERASE = 01h	Keine Datenbytes vorhanden

**CMD\_BLANK\_CHECK:** Es soll überprüft werden, ob der Flash-Speicher „leer“ ist

<b>00h:</b>	<b>01h:</b>	
Signatur-Byte: 'A' = 41h	Befehls-Byte: CMD_BLANK- _CHECK = 02h	Keine Datenbytes vorhanden

**CMD\_DEBUG:** Es sollen 64 Bytes aus dem Chip gelesen werden

<b>00h:</b>	<b>01h:</b>	
Signatur-Byte: 'A' = 41h	Befehls-Byte: CMD_DEBUG = 50h	Keine Datenbytes vorhanden

**CMD\_PROGRAM:** Eine Programmieranforderung wird gesendet

<b>00h:</b>	<b>01h:</b>	
Signatur-Byte: 'A' = 41h	Befehls-Byte: CMD- _PROGRAM = 03h	Keine Datenbytes vorhanden

## CMD\_HEXDATA: Ein Hex-Record soll in den Chip gebrannt werden

<b>00h:</b> Signatur-Byte: 'A' = 41h	<b>01h:</b> Befehls-Byte: CMD_HEXDATA = 40h	<b>02h:</b> LEN: Länge der folgenden Datenbytes	<b>03h:</b> ADRESSE: Hi-Byte der Startadresse im Flashspeicher	<b>04h:</b> ADRESSE: Low-Byte der Startadresse im Flashspeicher	<b>05h:</b> TYP: 00h = Daten 01h = EOF sonst. = Fehler	<b>06h:</b> ... <b>3Fh:</b> DATEN: Max. 58 Datenbytes, die ab der definierten Adresse in den Flash-Speicher gebrannt oder verifiziert werden sollen
--	--	---	--	---	--	---

## CMD\_VERIFY: Ein Hex-Record soll mit dem Flash-Speicher verifiziert werden

<b>00h:</b> Signatur-Byte: 'A' = 41h	<b>01h:</b> Befehls-Byte: CMD_VERIFY = 04h	Keine Datenbytes vorhanden
--	---	----------------------------

## CMD\_SET\_LOCK: Die Lockbits sollen gesetzt werden

<b>00h:</b> Signatur-Byte: 'A' = 41h	<b>01h:</b> Befehls-Byte: CMD_SET- _LOCK = 07h	<b>02h:</b> LOCKBITS: 01h = Lockbit 1 setzen 02h = Lockbit 1 und 2 setzen
--	---	--

## CMD\_GET\_VERSION: Die Version des Programmiergeräts soll ausgelesen werden

<b>00h:</b> Signatur-Byte: 'A' = 41h	<b>01h:</b> Befehls-Byte: CMD_GET- _VERSION = 20h	Keine Datenbytes vorhanden
--	--	----------------------------

## CMD\_GET\_TYPE: Der eingelegte Chiptyp soll ermittelt werden

<b>00h:</b> Signatur-Byte: 'A' = 41h	<b>01h:</b> Befehls-Byte: CMD_GET- _TYPE = 21h	Keine Datenbytes vorhanden
--	---	----------------------------

← Antworten vom USB-Programmer-Gerät:

**Allgemeine Bestätigung:** Wenn ein Befehl erfolgreich ausgeführt werden konnte, schickt das USB-Gerät ein Paket mit gültiger Signatur und dem Befehlsbyte=0 zurück (und ggf. angehängten Debugginginformationen).

**Fehler:** Wenn ein empfangenes Paket ungültig ist oder der Befehl zu einem Fehler führt, wird ein Paket mit gültiger Signatur, dem Befehlsbyte=CMD\_ERROR\_STR=30h und einem angehängten Fehlerstring zurückgeschickt.

**Debug-Antwort:** Erhält das USB-Gerät einen Debug-Befehl, werden die ersten 64 Bytes aus dem Flashspeicher ausgelesen und direkt als USB-Paket zurückgeschickt (ohne Signaturbyte etc.).

**Typ oder Versionsanfrage:** Nach einer Anfrage nach dem eingelegten Chiptyp oder der Programmerversion wird ein Paket mit gültiger Signatur zurückgeschickt.

- Für die Typanfrage gilt:  
Byte 03h enthält Manufacturer-ID  
Byte 04h enthält Product-ID
- Für die Versionsanfrage gilt:  
Ab Byte 03h steht die Versionsnummer als nullterminierter String

## 5.4 Das Intel-HEX-Format

Das Intel-Hex-Format ist im Gegensatz zur Binärdatei keine simple Abbildung des Flash-Speichers in eine Datei, in der Byte und Position 1-zu-1 übernommen werden können. Sondern es handelt sich dabei um eine Textdatei (meistens mit Endung .IHX oder .HEX), in der jede Zeile ein sog. Hex-Record in lesbarer Textform enthält. Ein Hex-Record definiert jeweils nur einen beliebigen, kleinen Teil des Speichers. Bei kleinen Programmen muss somit nicht der gesamte Flash-Speicher aktualisiert werden, sondern es reicht aus, nur die notwendigen Bytes, die im Hex-Record definiert wurden, zu überschreiben.

Die nachfolgenden Tabellen erklären die Reihenfolge und den Inhalt der Bytes in einem Hex-Record. Jeder Record enthält eine Zahl, die die Anzahl seiner Datenbytes angibt. Dann folgt die gewünschte Ablageadresse des ersten Datenbytes und zuletzt eine Checksumme. Eine solche Prüfsumme erhält man, wenn von Null die Summe aller Informationsbytes abgezogen wird und das Ergebnis auf das Low-Byte begrenzt wird. Start- und Endkennung zählen nicht zu den Informationsbytes. Eine komplette Intel-Hex-Datei enthält einen oder mehrere solcher Records. Die Datei endet mit einem End of File Record (EOF).

Die Informationsbytes und die Checksumme eines Intel-Hex-Formats werden nicht binär übertragen, sondern als ASCII codierte Ziffern. Start- und Endkennungen als ASCII-Zeichen. Die Binärzahl 6Ah wird dementsprechend als ‚6‘=36h und ‚A‘=41h übertragen. Die Checksumme wird vor dieser Zerlegung gebildet.

Bytes	Symbol	Beschreibung	Hex
1	:	Startkennung	3A
2	nn	Anzahl der Datenbytes im Record	
4	aaaa	Zieladresse des ersten Datenbytes im Record	
2	tt	Record-Typ: 00 Daten, 01 EOF, 02 erweitert	
n-2	dd	Datenbyte (Anzahl wie n angegeben)	
2	cc	Checksumme = Summe(nn..cc) exor 0xFF + 1	
1	CR	Endkennung Carriage Return	0D
1	LF	Endkennung Line Feed	0A

Beispiel: :02001300A30741  
:00000001FF

	nn	aaaa	tt	dd	cc	
:	02	0013	00	A3,07	41	CR,LF
Start	2 Datenbytes	Startadresse	Typ	Daten	Checksumme	Ende

Die Bitfolge 10100011 (A3h) wird in die 20. und 00000111 (07h) in die 21. Speicherstelle gebrannt.

	nn	aaaa	tt	dd	cc	
:	00	0000	01	-	FF	CR,LF
Start	0 Datenbytes	Startadresse	Typ	Daten	Checksumme	Ende

End of File Record

## 5.5 Die Firmware des Steuercontrollers AT89C5131

Der Controller AT89C5131 stellt das Herzstück der Schaltung dar. Die Firmware ermöglicht die Steuerung des Programmiers über den USB-Bus (Implementation der Kommandos, die im vorigen Kapitel genannt wurden) und stellt verschiedene Signale und Datenbytes auf der Hardwareseite für den Flashvorgang bereit.

### 5.5.1 Projektgliederung und der Kompilationsvorgang

Die Firmware für den Atmel Controller ist in der Sprache C geschrieben und kann mit Hilfe des SDCC Compilers übersetzt werden (Aufruf von „*go.bat*“). Dabei wird eine Hex-Datei erzeugt, die den Inhalt des Flashspeichers (das kompilierte Programm) enthält. Diese kann wie in Kapitel „5.2 Einspielen der AT89C5131-Firmware mittels *ausbprog.exe*“ beschrieben eingespielt werden.

Zum Projekt gehören die folgenden Quelldateien (eine Projektdatei gibt es nicht):

<b>Headerdateien:</b>	
at89c5131.h	Definiert die Ports und SFR-Register des AT89C5131 und ist an die Syntax des SDCC-Compilers angepasst (abgeänderte Version von Atmels Beispiel-Headerdatei)
commands.h	Definiert Konstanten und Werte für die programminternen USB-Befehle zur Kommunikation zwischen Firmware und dem Windows Programm
compiler.h	Atmel-Headerdatei mit speziellen Defines zur Anpassung an den Compiler
ext_5131.h	Atmel-Headerdatei für den 5131-Controller. Definiert Konstanten für Interrupts und Register/Flags
flash.h	Definiert Konstanten für die LEDs und enthält Deklarationen für das Modul „flash.c“
usb.h	Definiert USB-relevante Typen und Konstanten, insbesondere für die Enumeration. Enthält Deklaration für das Modul „usb.c“
<b>Quelldateien:</b>	
main.c	Enthält den Programmeinstiegspunkt (main). In der Hauptroutine werden Variablen und die USB-Einheit initialisiert, anschließend läuft der Controller in einer Endlosschleife. Die eigentlichen Routinen werden durch Interrupts aufgerufen.
flash.c	Enthält Routinen zur Auswertung eines empfangenen USB-Pakets und ist für sämtliche Flashvorgänge zuständig wie z.B. Löschen, Schreiben, Auslesen, Chiptyp ermitteln etc.
usb.c	Beinhaltet sämtliche USB-Funktionen, z.B. Funktionen zur Initialisierung und Konfiguration der Endpoints und PLL (auf 48MHz), sowie USB-Interrupt-Funktionalität, Bulk-Transfers und Enumeration
<b>Sonstige Dateien:</b>	
go.bat	Diese Windows Batch-Datei vereinfacht das Kompilieren und Linken der drei C-Quelldateien. Ein Aufruf (z.B. Doppelklick) ruft den SDCC zum Kompilieren auf und erzeugt im Unterverzeichnis „output“ eine gepackte Intel-Hex-Datei.
output-Verzeichnis	Dieses Unterverzeichnis enthält die vom SDCC bei der Kompilierung erzeugten Dateien. Das sind zum Beispiel Assemblerlistings (.ASM) und Symboldateien (.SYM), sowie die Hex-Dateien (.IHX und .HEX).

## 5.5.2 Problemanalyse und grundsätzlicher Programmaufbau

Die folgende Abbildung zeigt den grundsätzlichen Programmaufbau der Firmware:

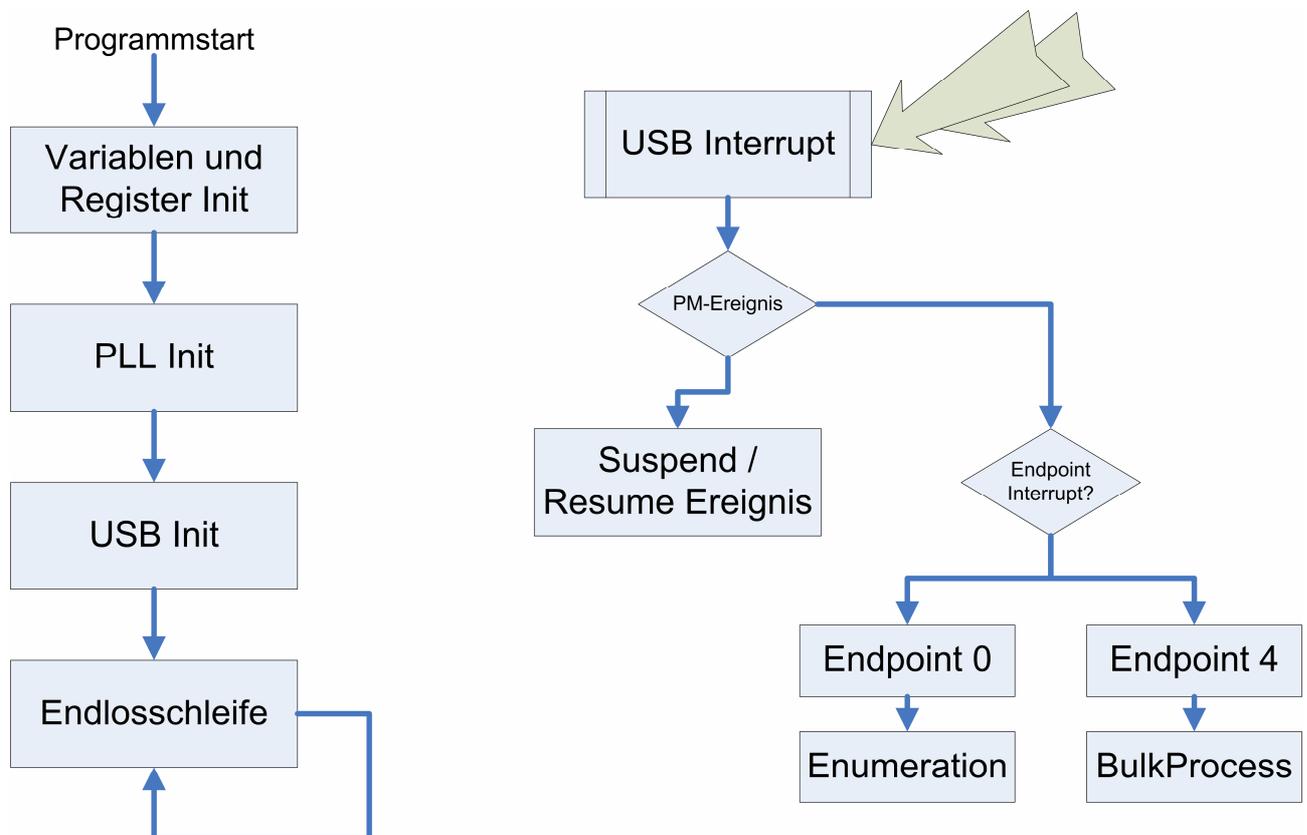


Abbildung 5-3: Die Firmware

### Programmstart (main.c):

- Initialisierung der Variablen und Register
- Einstellen der PLL des AT89C5131-Controllers auf 48 MHz (für USB benötigt)  
„usb\_init\_pll();“
- Initialisieren der USB-Register und Konfiguration der Endpoints im Controller  
„usb\_init();“
- Freigeben der Interrupts (für USB)
- Zum Schluss läuft der Mikrocontroller in einer Endlosschleife

### USB-Kommunikation (usb.c):

- Wenn ein **USB-Interrupt** auftritt wird die ISR „usb\_interrupt\_process()“ aufgerufen. Dort wird zunächst zwischen den Endpoints 0 und 4 unterschieden. Gelangt ein Paket auf Endpoint 0 (für Control-Transfer konfiguriert) kann es sich nur um eine Anfrage während des Enumeration-Prozesses handeln, die weitere Verarbeitung erfolgt durch Aufruf der Funktion „usb\_enumeration\_process()“. Hat ein Paket im Endpoint 4 den Interrupt ausgelöst, so handelt es sich um Bulk-Daten. Die eigentliche Kommunikation zwischen der Hardware des USB-Programmers und dem Windows-Steuerprogramm erfolgt im Bulk-Transfer, entsprechend wird „usb\_bulk\_process()“ aufgerufen.
- Die USB-Anmeldung beim Betriebssystem und das Aushandeln der Konfigurationsdaten (**Enumeration**) wird im nachfolgenden Kapitel beschrieben.

- Wird ein Bulk-Paket empfangen, so werden zunächst alle Bytes in ein globales Puffer-Array „*g\_buff[]*“ gespeichert (dies geschieht in der Funktion „*usb\_bulk\_process()*“), anschließend erfolgt die Untersuchung und Verarbeitung des Pakets durch Aufruf von „*usb\_eval()*“ (in *flash.c*).
- Um ein **Paket zum PC senden** zu können, muss zunächst der gewünschte Endpoint ausgewählt werden (0=Control-Transfer oder 5=Bulk-Transfer). Dann können die einzelnen Datenbytes in das interne Schieberegister gespeichert und anschließend durch Setzen des TX-Ready-Bits übertragen werden (→ z.B. in der Funktion „*usb\_send\_data()*“).

### Implementation der Flash-Vorgänge (*flash.c*):

- In der Funktion „*usb\_eval()*“ wird eine Anforderung vom PC untersucht. Dazu wird das Paket in die einzelnen Bytes zerlegt. Stimmt die Signaturkennung, kann das Befehlsbyte ausgewertet und einem Befehl wie z.B. „Typ ermitteln“ oder „Flash-Löschen“ zugeordnet werden. Anschließend wird die entsprechende Routine aufgerufen (→ siehe folgende Kapitel).
- Die einzelnen Routinen für die Flash-Operationen beginnen jeweils mit „*chip\_*“ (z.B. „*chip\_blank\_check()*“ oder „*chip\_erase()*“)
- Der Reset-Pin des eingelegten AT89C2051/4051 muss für der Programmierung auf die drei Spannungen 0V, +5V und +12V gesetzt werden können. Das wird in den Funktionen „*set\_reset\_0v()*“, „*set\_reset\_5v()*“ und „*set\_reset\_12v()*“ implementiert, indem die Ausgangspins des Controllers entsprechend gesetzt werden.

## 5.5.3 Der USB-Enumerationsprozess

Bevor das Windows-Programm mit dem USB-Gerät kommunizieren kann, muss der PC das Gerät erkennen und den richtigen Gerätetreiber zuordnen. Dazu findet nach dem Einstecken des USB-Geräts ein anfänglicher Datenaustausch (Enumerationsprozess) mit dem PC statt, der im folgenden Kapitel genauer beschrieben wird.

### 5.5.3.1 Der Ablauf der Enumeration

Die nachfolgende Übersicht zeigt die typischen Schritte während der Enumeration unter Windows. Dabei ist zu beachten, dass die Reihenfolge in der die Anforderungen und Ereignisse auftreten nicht festgelegt ist! Die Firmware muss also alle Anforderungen zu jedem Zeitpunkt erkennen und bearbeiten können; der folgende Ablauf ist nur ein Beispiel:

1. Anwender verbindet das Gerät mit dem USB-Port (oder der Rechner wird eingeschaltet)
2. Der USB-HUB erkennt das Gerät und ordnet es einer Geschwindigkeitsklasse zu
 

Low-Speed:	Gerät hat 1,5k-Pullup-Widerstand an D- Leitung
Full/High-Speed:	Gerät hat 1,5k-Pullup-Widerstand an D+ Leitung
3. Der USB-HUB löst einen Interrupt aus, um den PC über das angeschlossene Gerät zu benachrichtigen
4. Der USB-HUB setzt das Gerät zurück (Reset/Null-Impuls auf den Datenleitungen für mind. 10ms)
5. Der USB-HUB stellt einen Signalpfad zwischen dem Gerät und dem Bus her (PC sendet „GET\_PORT\_STATUS“-Anforderung)

6. Der PC ermittelt die maximale Paketlänge des Übertragungskanals (PC sendet „GET\_DESCRIPTOR“-Anforderung, liest aber nur die max. Paketlänge aus der Antwort)
7. Der PC ordnet dem USB-Gerät eine neue, eindeutige Adresse zu. Die bisherige Kommunikation verlief immer über die Standardgeräteadresse 0, Endpunkt 0 (PC sendet „SET\_ADDRESS“-Anforderung)
8. Der PC ermittelt die Fähigkeiten des USB-Geräts (PC sendet erneut die „GET\_DESCRIPTOR“-Anforderung, liest aber diesmal die gesamte Antwort)
9. Der PC ordnet dem USB-Gerät einen Gerätetreiber zu und lädt ihn (Der Treiber wird anhand von Anbieter-, Produkt-ID, Release-Nummer und Klassen-ID zugeordnet. Kann kein passender Treiber gefunden werden, wird der Anwender zur Installation eines neuen Treibers aufgefordert)

Die Umsetzung der Enumeration in der Firmware des USB-Programmers ist in der Quelldatei „usb.c“ beschrieben. In der Interruptserviceroutine „usb\_interrupt\_process()“ werden die Setup-Requests auf Endpunkt 0 (Control-Transfer) erkannt und durch Aufruf der Funktion „usb\_enumeration\_process()“ abgearbeitet.

### 5.5.3.2 Die wichtigsten Deskriptoren

Der Informationsaustausch zwischen PC und dem USB-Gerät erfolgt mit Hilfe spezieller, fest definierter Datenstrukturen, den so genannten Deskriptoren.

Das USB-Gerät muss zum einen unterschiedliche Anforderungen vom PC empfangen, zum anderen müssen entsprechende Antwort-Deskriptoren bereitgestellt und zum PC verschickt werden.

Die Firmware dieses USB-Programmers kann nach entsprechender Anfrage die folgenden Deskriptoren zum PC senden:

Wert (Hex)	Deskriptor	Beschreibung
0x01	DEVICE	Enthält Informationen über das USB-Gerät als Ganzes und die Anzahl der unterstützten Konfigurationen.
0x02	CONFIGURATION	Enthält Informationen über den Strombedarf und die Anzahl der unterstützten Schnittstellen
0x04	INTERFACE	Enthält Informationen über die vom Gerät unterstützten Endpunkte.
0x05	ENDPOINT	Enthält für die Kommunikation wichtige Informationen über einen Endpunkt.
0x03	STRING (LANGUAGE)	Enthält optionalen Text: Sprach-ID
0x03	STRING (MANUFACTURER)	Enthält optionalen Text: Anbieter Name
0x03	STRING (PRODUCT)	Enthält optionalen Text: Produkt Name
0x03	STRING (SERIAL NUMBER)	Enthält optionalen Text: Seriennummer

Die entsprechenden Typen (structs) werden in der Headerdatei „usb.h“ definiert. In der Quelldatei „usb.c“ werden die Datenstrukturen initialisiert und verwendet!

### 5.5.3.3 Die wichtigsten Befehlsanforderungen (USB-Standard-Requests)

Der USB-Programmer muss (wie jedes USB-Gerät) auf die elf untenstehenden Anforderungen vom PC, USB-Standard-Requests genannt, reagieren. Die Setup-Pakete werden vom PC auf der Control-Transfer-Ebene an Endpoint 0 geschickt, die Firmware des USB-Geräts ist dann für die korrekte Verarbeitung zuständig.

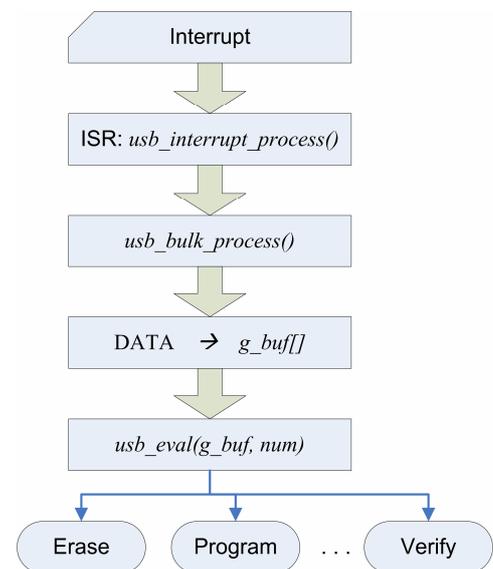
Die Auswertung und Verzweigung einer Anfrage wird über die Funktion „*usb\_enumeration\_process()*“ in der Datei „*usb.c*“ realisiert. Dort wird je nach Anforderung vom PC eine eigene Unterfunktion aufgerufen, die alle gewünschten Informationen ermittelt und entsprechende Antworten zurücksendet. Die folgende Tabelle zeigt, welche Anforderungen vom PC (Requests) möglich sind:

Nr. (Hex)	Request	Beschreibung
0x00	GET_STATUS	PC fordert Statusinformationen an (Geräte-, Schnittstellen- oder Endpunktstatus)
0x01	CLEAR_FEATURE	PC fordert das Rücksetzen eines der beiden Features: DEVICE_REMOTE_WAKEUP oder ENDPOINT_HALT
0x03	SET_FEATURE	PC fordert das Setzen eines Features
0x05	SET_ADDRESS	PC gibt eine Adresse für die Kommunikation mit dem USB-Gerät vor
0x06	GET_DESCRIPTOR	PC fordert das USB-Gerät zum Senden eines bestimmten Deskriptors auf. (→ siehe voriges Kapitel)
0x08	GET_CONFIGURATION	PC fordert die aktuelle Konfiguration an
0x09	SET_CONFIGURATION	PC fordert das USB-Gerät auf, eine bestimmte Konfiguration zu verwenden
0x0A	GET_INTERFACE	PC fordert aktuelle Schnittstelleneinstellung an
0x07 0x0B 0x0C	SET_DESCRIPTOR SET_INTERFACE SYNCH_FRAME	Auf diese Anforderungen reagiert der USB-Programmer lediglich mit einem STALL

### 5.5.4 Empfang und Auswertung der USB-BULK-Pakete

Wenn das Gerät ein USB-Paket empfängt, wird ein Interrupt ausgelöst und die Interrupt-Service-Routine „*usb\_interrupt\_process()*“ aufgerufen. Diese Routine überprüft, um was für ein Paket es sich handelt und an welchen Endpoint im Controller das Paket gerichtet ist. Handelt es sich um ein Bulk-Paket („*Usb\_bulk\_received()*“ liefert True), so wird die Funktion „*usb\_bulk\_process()*“ für die weitere Verarbeitung aufgerufen. Diese Funktion wiederum liest alle ankommenden Daten in das globale Puffer-Array „*g\_buf[]*“ ein.

Anschließend kann der Puffer analysiert und der vom PC ausgehende Befehl ermittelt und verarbeitet werden. Dazu wird „*usb\_eval(g\_buf, num)*“ aufgerufen. Diese Funktion prüft zunächst die Mindestlänge (PACKET\_HEADER\_LEN) des Pakets.



Ebenso muss das Signaturbyte (erstes Byte) gültig sein (=PACKET\_SIG). Andernfalls ist der Befehl vom PC bedeutungslos und eine Fehlermeldung wird zurückgeschickt.

Anhand des zweiten Bytes wird der eigentliche Befehl decodiert. Die möglichen Werte bzw. die Zuordnungen werden durch die Defines „*CMD\_???*“ in der Headerdatei „*commands.h*“ abgebildet. Je nach Befehl wird in eine entsprechende Unterroutine verzweigt. Besitzt das Befehlsbyte beispielsweise den Wert „*CMD\_ERASE*“, so wird die Funktion „*chip\_erase()*“ aufgerufen, bei „*CMD\_BLANK\_CHECK*“ entsprechend „*chip\_blank\_check(g\_flash\_size)*“ usw.

### 5.5.5 Signaturbytes vom AT89C2051/4051 auslesen

Wie im Datenblatt zum AT89C2051/4051 zu finden ist, können die beiden Signaturbytes des Chips relativ einfach durch Anlegen entsprechender Spannungen an den Steuerpins und das Einhalten der richtigen Timings über den Datenbus P1 ausgelesen werden. Die Funktion „*chip\_read\_signature()*“ der Firmware des USB-Programmers übernimmt genau diese Aufgabe:

Zu Beginn wird der eingelegte Chip initialisiert. Dazu wird die Funktion „*chip\_init()*“ (in „*flash.c*“) aufgerufen, die die im Datenblatt beschriebene Power-Up-Sequence durchführt, um den Adresszähler zurückzusetzen und das Flash einzuleiten. Dabei wird das in der Abbildung rechts zu sehende Timing-Verhalten nachgebildet.

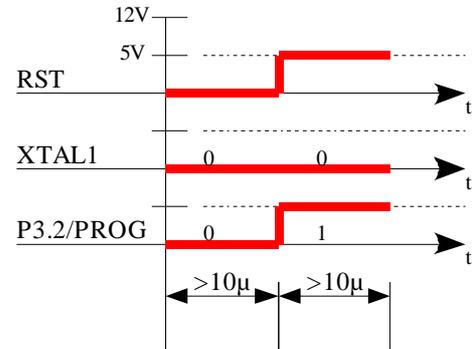


Abbildung 5-4: Power-Up-Timing - *chip\_init()*

Nachdem der Chip initialisiert wurde, werden die vier Steuerpins P3.3, P3.4, P3.5 und P3.7 entsprechend der Tabelle „Flash Programming Modes“ auf Low gelegt.

Anschließend kann das erste Signaturbyte am Port P1 eingelesen und abgespeichert werden.

Durch Aufruf der Funktion „*inc\_addr()*“ wird ein Impuls am Adresszähler (XTAL1-Pin) erzeugt, um zur nächsten Adresse im Flashspeicher zu gelangen.

Dann kann das zweite Signaturbyte eingelesen werden.

Das erste Signaturbyte enthält einen Identifier für den Herstellernamen des Chips, das zweite Byte steht für den Chiptypen. Da die verschiedenen Chiptypen unterschiedlich große Flash-Speicher besitzen, wird die jeweils aktuelle Größe abgespeichert.

Die untenstehende Abbildung verdeutlicht noch einmal das Timing-Verhalten:

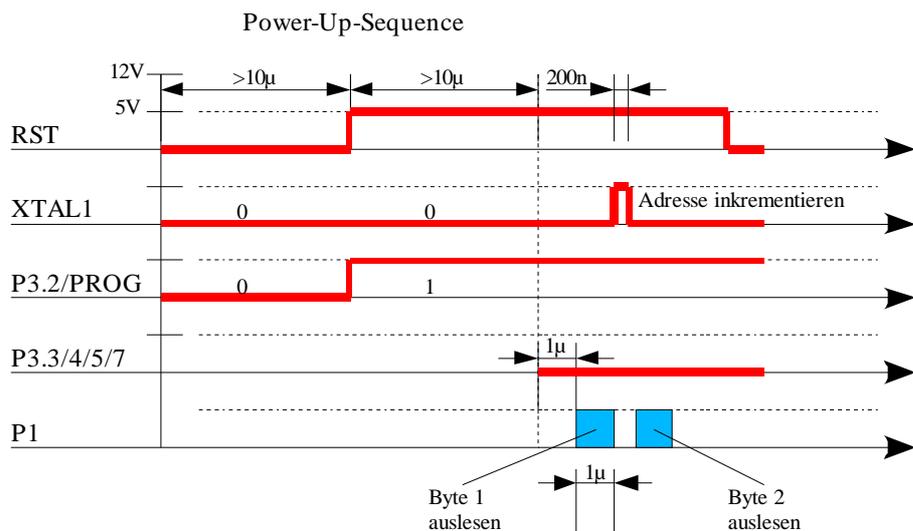


Abbildung 5-5: Timing-Verhalten beim Auslesen der Signaturbytes

## 5.5.6 Löschen des Flash-Speichers vom AT89C2051/4051

Das Löschen des eingelegten Chips wird durch Aufruf der Funktion „*chip\_erase()*“ (aus der Quelldatei „*flash.c*“) erreicht. Die Funktion initialisiert den Chip (Power-Up-Sequence) und legt die entsprechenden Pegel aus der Tabelle „Flash Programming Modes“ an den Steuerpins an:

P3.3	P3.4	P3.5	P3.7
High	Low	Low	Low

Der Reset-Eingang wird wie beim Schreibzugriff üblich auf 12V geschaltet. Der Flash-Speicher wird nun komplett gelöscht, indem der Pin P3.2/PROG für mindestens 10ms auf Low bleibt.

## 5.5.7 Auslesen und Schreiben des Flash-Speichers vom AT89C2051/4051

Das Schreiben und Auslesen (Verifizieren) des Flash-Speichers läuft sehr ähnlich ab und wird deshalb gemeinsam betrachtet. Insgesamt stehen dem PC die drei Befehle

- CMD\_PROGRAM
- CMD\_VERIFY
- CMD\_HEXDATA

zur Verfügung. Zu Beginn wird einer der ersten beiden Befehle zum Gerät gesendet, um das Auslesen (Verifizieren) oder Schreiben einzuleiten. Anschließend können durch das wiederholte Senden des dritten Befehls (CMD\_HEXDATA) die eigentlichen Daten übertragen werden. In der Funktion „*usb\_eval(..)*“ (in der Quelldatei „*flash.c*“) werden die Befehle decodiert und entsprechende Routinen zur Verarbeitung aufgerufen.

Wie die rechte Abbildung verdeutlicht, sendet der PC zu Beginn einen Befehl (CMD\_PROGRAM oder CMD\_VERIFY) zum Gerät. Der Befehl wird (in „*usb\_eval(..)*“) erkannt und „*chip\_init\_burn\_or\_verify(..)*“ wird aufgerufen. Diese Funktion initialisiert die Steuer-Ports und schaltet den Reset-Pin auf die benötigte Spannung (12V=brennen, 5V=auslesen). Außerdem wird der Adresszähler zurückgesetzt und eine globale Variable speichert, ob gelesen oder geschrieben werden soll (notwendig für die folgenden CMD\_HEXDATA-Befehle!). Anschließend wird eine Bestätigung via USB an den PC zurückgeschickt, die eigentlichen Daten, die geschrieben bzw. verifiziert werden sollen, können folgen.

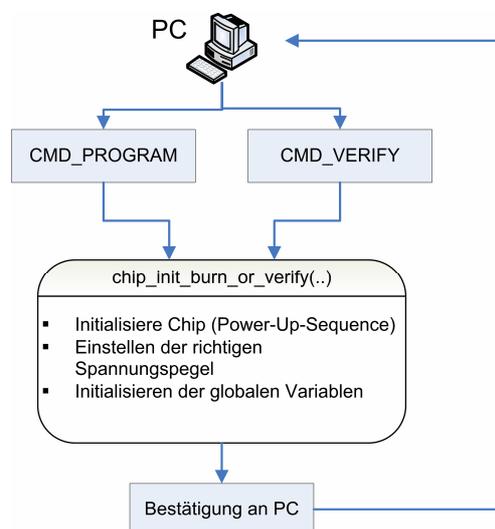


Abbildung 5-6: PC sendet Write-/Verify-Anfrage

Nach der Initialisierung schickt der PC die eigentlichen Daten der Hex-Records, die entweder in den eingelegten Chip geschrieben oder mit den aus dem Chip stammenden Daten verglichen werden. Der Aufbau des Hexdata-Pakets wurde bereits in Kapitel „5.3.2 Aufbau der verschickten Pakete“ beschrieben.

Die Funktion „*chip\_burn\_or\_verify(..)*“ überprüft zunächst die einzelnen Bytes, ermittelt die aktuelle Adresse aus dem Hex-Record und geht bis zu dieser Adresse im Chip (durch Inkrementieren des Adresszählers: Impulse am XTAL1-Pin). In einer Schleife werden dann die einzelnen Bytes durch Anlegen entsprechender Pegel und Einhalten des Timings in den Chip geschrieben bzw. mit den alten Daten verglichen. Wurde ein Paket (Hex-Record) abgearbeitet wird eine Bestätigung zum PC gesendet und das nächste Paket kann folgen. Dieser Vorgang wird so lange wiederholt, bis alle gewünschten Bytes im Flash-Speicher durchlaufen wurden.

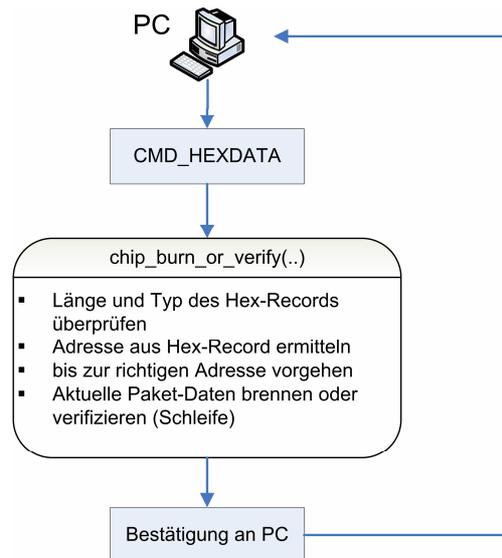


Abbildung 5-7: PC sendet Hex-Daten

### 5.5.8 Setzen der Lockbits vom AT89C2051/4051

Das Schreiben der Lockbits übernimmt die Funktion „*chip\_write\_lockbits(..)*“ aus der Quelldatei „*flash.c*“. Nach dem Initialisieren des Chips (Power-Up-Sequence) wird zu Beginn die Versorgungsspannung am Reseteingang auf 12V geschaltet. Anschließend werden die Steuerpins für das erste Lockbit entsprechend der Tabelle „Flash Programming Modes“ gesetzt:

P3.3	P3.4	P3.5	P3.7
High	High	High	High

Ein Impuls am Port P3.2/PROG leitet daraufhin den Schreibvorgang ein. Mittels „Data Polling“ am Pin 3.1/(RDY/BSY) kann abgefragt werden, wann der Schreibvorgang erfolgreich abgeschlossen wurde.

Das zweite Lockbit kann optional auf dem gleichen Wege gesetzt werden, die Steuerpins müssen lediglich wie folgt geändert werden:

P3.3	P3.4	P3.5	P3.7
High	High	Low	Low

## 5.6 Das Windows-GUI-Programm

Die Windowsanwendung bietet dem Anwender die komfortable Möglichkeit in einer grafischen Oberfläche mit dem USB-Programmer zu kommunizieren und Befehle abzusetzen. Der Aufbau des Programms und die Programmierung wird in diesem Kapitel näher erläutert.

## 5.6.1 Projektübersicht und Kompilierung

Das Windows-GUI-Programm „*usb\_programmer.exe*“ wurde mit Microsoft Visual C++ 6.0 geschrieben, allerdings ohne Verwendung der Klassenbibliothek MFC. Die Quell- und Projektdateien befinden sich im Verzeichnis „\Quellentexte\Windows GUI (VISUAL C)“ auf der beiliegenden CD-ROM. Die untenstehende Tabelle gibt eine Übersicht über die wichtigsten zum Projekt gehörenden Dateien:

<b>Headerdateien:</b>	
log.h	Header-Datei für log.cpp.
commands.h	Header-Datei für commands.cpp. Definiert Konstanten und Typen für die programminternen USB-Befehle zur Kommunikation zwischen Firmware und dem Windows Programm
resource.h	Automatisch generierte Headerdatei, die von Visual Studio verwaltet wird und sämtliche Konstanten (IDs) für die Ressourcen beinhaltet.
usb.h	Header-Datei für usb.cpp.
<b>Quelldateien:</b>	
main.cpp	Enthält den Programmeinstiegspunkt (WinMain). In der Hauptroutine wird das Hauptfenster erzeugt und in die Windows Message Loop eingeklinkt. Enthält außerdem die Callback-Funktion des Hauptfensters sowie die Thread-Funktionen, Routinen zum Laden der Hexdateien und andere.
commands.cpp	Enthält Funktionen zum Abschicken und Empfangen der Programmierbefehle
log.cpp	Enthält Funktionen für die Ausgabe im Log-Fenster der Anwendung
usb.cpp	Dieses Modul stellt Funktionen für die direkte Kommunikation mit dem USB-Treiber bereit
<b>Sonstige Dateien:</b>	
icon1.ico	Programm-Icon
res.rc	Resource Script der Anwendung. Enthält Dialogvorlage, das Hauptmenü und eine Verknüpfung zum Programm-Icon
usb_programmer.dsp	Visual C++ 6.0 Projektdatei
usb_programmer.dsw	Visual C++ 6.0 Workspace – Über diese Datei kann das gesamte Projekt bequem geladen werden!
Release-Verzeichnis	In diesem Unterverzeichnis wird das fertige Programm (Exe-Datei) nach dem Kompilieren/Linken gespeichert.
Debug-Verzeichnis	In diesem Unterverzeichnis wird die Debug-Version des Programms (Exe-Datei mit integrierten Debuginformationen) angelegt.

Das Projekt kann durch Öffnen der Datei „*usb\_programmer.dsw*“ in die IDE von Visual Studio geladen werden. Im Arbeitsbereich ist dann der schnelle Zugriff auf alle Projektdateien möglich. Über den Menüpunkt „Erstellen → Alles neu erstellen“ oder über die entsprechenden Buttons in der Toolbar wird die Anwendung kompiliert und erzeugt (Je nach Auswahl Debug- oder Release-Version).

## 5.6.2 Erstellung und Behandlung der Programmoberfläche

In der Eintrittsfunktion „*WinMain(..)*“ (in „*main.cpp*“) wird über die Windows API-Funktion „*CreateDialog(..)*“ das Hauptfenster aus der Dialogresource erzeugt. Der eigentliche Aufbau der Oberfläche (wie die Buttons angeordnet sind etc.) ist in der Resourcendatei „*res.rc*“ abgespeichert und kann jederzeit über den Resource-Editor im Visual Studio bearbeitet werden.

Als nächstes folgt in der Eintrittsfunktion die Ereignis-Warteschleife des Programms, mit deren Hilfe das Programm die Nachrichten von Windows verarbeiten kann:

```

while (GetMessage(&msg, NULL, 0, 0))
{
    if (g_hMainWnd==0 || !IsDialogMessage(g_hMainWnd, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

```

Die Callback-Routine „*MainWndProc(..)*“ ist direkt dem Dialogfenster zugeordnet und wird jedes Mal aufgerufen, wenn ein Ereignis im Hauptfenster auftritt (z.B. Button gedrückt, Maus bewegt...). Das Ereignis wird abgefragt und einer bestimmten Routine zugeordnet. So wird für jeden Button eine entsprechende Funktion zur Verarbeitung aufgerufen.

### 5.6.3 Laden und Verarbeiten neuer Hex- und Binärdateien

Wenn der „*Durchsuchen-Button*“  im Hauptfenster gedrückt wird, kann eine Hex- oder Binärdatei ausgewählt werden, die anschließend geladen wird. Durch Klick auf den Button wird die Funktion „*LoadHexFile(..)*“ aufgerufen, die einen Dateiauswahldialog anzeigt und nach der Wahl den selektierten Dateinamen speichert. Es wird versucht, die Datei zu Öffnen (`CreateFile(filename, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL)`) und anhand der Extension wird zwischen Hex- und Binärformat unterschieden:

HEX	BIN
Extension: .hex oder .ihx	Beliebige andere Extension, aber Dateigröße darf 4096 Byte nicht überschreiten!

Handelt es sich um eine Hex-Datei, wird zunächst überprüft, ob das Intel-Hex-Format korrekt eingehalten wurde. Dazu liest die Funktion „*CheckHexFile(..)*“ die gesamte Datei zeilenweise ein und prüft die einzelnen Bytes der Hex-Records. Ist alles in Ordnung, werden Flashgröße und die letzte Speicheradresse mittels „*SetDlgItemText(..)*“ angezeigt.

Soll der eingelegte Chip später beschrieben bzw. verifiziert werden, müssen die Daten aus der Hex- oder Binärdatei zum Gerät geschickt werden. Aufgrund überlappender Bereiche (mehrfache Nennung bestimmter Speicheradressen...) und der Tatsache, dass die Speicherbereiche laut Intel-Hex-Standard nicht sortiert sein müssen, wird sowohl aus den Hex-Records als auch der Binärdatei vor Beginn der Übertragung ein Speicherabbild erstellt. Dies geschieht im Quellcode in der Routine „*chip\_program(..)*“ (in „*commands.cpp*“) durch Aufruf der Funktion:

```

read_hexfile(hexfile, &m, C_MAX_FLASH_SIZE) bzw.
read_binfile(hexfile, &m, C_MAX_FLASH_SIZE)

```

Das Speicherabbild wird durch folgenden Typen repräsentiert:

```

typedef struct st_mem
{
    unsigned char data[C_MAX_FLASH_SIZE];
    BOOL enable[C_MAX_FLASH_SIZE];
} t_mem;

```

Die Abbildung 5-8 verdeutlicht den Aufbau des Speicherabbilds. Es besteht aus zwei Arrays in der Größe des max. Flash-Speichers. Das Datenfeld enthält für jede Adresse das zugeordnete Datenbyte, das entsprechende Enable-Flag gibt an, ob die Adresse im Flashspeicher des eingelegten Chips verarbeitet werden soll (enable=True) oder ob die Adresse übersprungen werden soll (enable=False, der alte Werte bleibt erhalten).

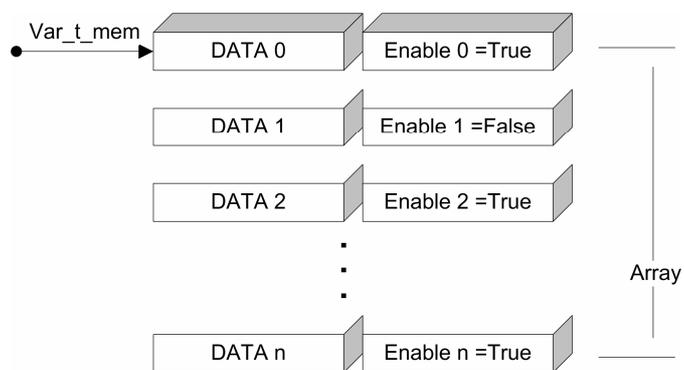


Abbildung 5-8: Das Speicherabbild

Um die einzelnen Datenbytes an den USB-Programmer zu senden, werden die gültigen Bytes aus dem Speicherabbild vor der Übertragung wieder in kleine USB-gerechte Pakete (ähnlich den Hex-Records) aufgeteilt. Diese Pakete werden durch Variablen vom Typ „t\_packet“ dargestellt:

```
typedef struct st_packet
{
    unsigned char    sig;
    unsigned char    command;
    unsigned char    data[PACKET_SIZE-2];
    size_t           len;
} t_packet;
```

Der Umweg über das Speicherabbild hat den Vorteil, dass auch unsortierte Hex-Dateien mit Speicherüberlappungen verwendet werden können (wie sie z.B. der SDCC erzeugt).

### 5.6.4 Reaktion auf Benutzereingaben, Ausführung in Threads

Wird ein Befehl an das USB-Gerät gesendet, so kann die Verarbeitung je nach Befehl und Anzahl der Datenbytes einige Zeit in Anspruch nehmen. Um das Programm für diesen Zeitraum nicht zu blockieren, werden potenziell längere Befehle in eigenen Threads ausgeführt, die von Windows separat verwaltet werden und parallel zum Hauptprogramm ablaufen. Einfache Befehle werden direkt bearbeitet, da keine lange Verzögerungsdauer entstehen kann.

Folgende komplexere Befehle werden in Threads ausgeführt:

<b>Befehl:</b>	<b>Thread-Funktion:</b>
Programmieren (CMD_PROGRAM)	ProgramThread(PVOID pvoid)
Verifizieren (CMD_VERIFY)	VerifyThread(PVOID pvoid)
Automatisches Programmieren (Erase, Blank Check, Program und Verify)	AutoThread(PVOID pvoid)

Die Threads werden unter Windows mit dem Befehl „CreateThread(..)“ bzw. dem simpleren Makro „\_beginthread(ThreadProg, 0, NULL)“ erzeugt und gestartet.

Das Konsolenprogramm verwendet keine Threads!

## 5.6.5 USB-Kommunikation – Senden von Befehlen

In der Quelldatei „*commands.cpp*“ sind die folgenden sechs Funktionen enthalten, die die unterschiedlichen Befehle bzw. Anfragen an das USB-Gerät senden:

chip_blank_check(..)	Führt einen Blank-Check nach dem Löschen durch
chip_debug(..)	Debug-Funktion, liest 64 Bytes aus dem Chip aus
chip_erase(..)	Löscht den Flashspeicher des Chips vollständig
chip_program(..)	Sendet Hexdaten an den USB-Programmer um den Chip zu Programmieren oder zur Verifizierung
chip_set_lockbits(..)	Setz die Lockbits des Chips
chip_type_and_version(..)	Ermittelt den eingelegten Chip-Typ und die Version des USB-Programmers

Alle Funktionen rufen für die eigentliche Übertragung eines USB-Pakets die Funktion „*send\_packet(t\_packet \*d\_in, t\_packet \*d\_out)*“ (ebenfalls aus „*command.cpp*“) auf. Die Funktion überprüft und sendet das durch „*d\_out*“ angegebene Paket zum USB-Controller und empfängt automatisch eine Bestätigung. Das empfangene Paket wird ausgewertet und steht (wenn kein Fehler vorliegt) in der durch „*d\_in*“ angegebene Struktur zur Weiterverarbeitung bereit. Für die eigentliche USB-Übertragung werden die in „*usb.cpp*“ enthaltenen elementaren Funktionen „*usb\_init()*, „*usb\_send\_bulk()*, „*usb\_get\_bulk* und „*usb\_free()*“ benutzt. Diese kommunizieren direkt mit dem USB-Gerätetreiber und werden in Kapitel „5.8 Der eingesetzte USB-Treiber“ genauer erläutert. Wie der untenstehende Beispielausschnitt aus dem Quelltext von „*chip\_erase(..)*“ zeigt, muss vor dem Aufruf der Funktion „*send\_packet(..)*“ zunächst der Parameter „*d\_out*“ initialisiert werden:

```
BOOL chip_erase(void)
{
    t_packet    d_in, d_out;

    Log("--- Sende Befehl zum Löschen des Chips...\r\n");
    ZeroMemory (&d_out, sizeof(d_out) );
    d_out.sig      = PACKET_SIG;
    d_out.command  = CMD_ERASE;
    d_out.len      = 0;
    if (!send_packet(&d_in, &d_out) )
    {
        Log("\r\nERR: Konnte Chip nicht löschen!\r\n");
        return FALSE;
    }

    return TRUE;
}
```

Zunächst werden alle Bytes der lokalen Variable „*d\_out*“ zurückgesetzt (durch „*ZeroMemory(..)*“). Dann wird die Paketsignatur gespeichert, bei ungültiger Signatur würde das USB-Gerät den Befehl verweigern. Anschließend wird der durch die Konstante „*CMD\_ERASE*“ codierte Befehl eingestellt. Da für diesen Befehl keine Daten benötigt werden, wird schließlich noch die Länge der Datenbytes auf Null gesetzt (*d\_out.len* = 0).

## 5.7 Das Windows-Kommandozeilenprogramm

### 5.7.1 Projektübersicht und Kompilierung

Das Windows-Konsolenprogramm „*usb\_programmer.exe*“ wurde ebenfalls mit Microsoft Visual C++ 6.0 geschrieben. Die Quell- und Projektdateien befinden sich im Verzeichnis „\Quelltexte\Windows Konsole (VISUAL C)“ auf der beiliegenden CD-ROM. Der Projektaufbau ist der GUI-Version relativ ähnlich. Die Eintrittsfunktion „*main(..)*“ befindet sich jedoch in der Quelldatei „*usb\_programmer.cpp*“. Da kein Hauptfenster erstellt und verarbeitet werden muss, entfallen viele Funktionen der GUI-Version.

Das Projekt kann durch Öffnen der Datei „*usb\_programmer.dsw*“ in die IDE von Visual Studio geladen werden. Im Arbeitsbereich ist dann der schnelle Zugriff auf alle Projektdateien möglich. Über den Menüpunkt „Erstellen → Alles neu erstellen“ oder über die entsprechenden Buttons in der Toolbar wird die Anwendung kompiliert und erzeugt (Je nach Auswahl Debug- oder Release-Version).

### 5.7.2 Die Kommandozeile

Im Gegensatz zum grafischen Windows-Programm lautet die Eintrittsfunktion bei der Konsolenanwendung nicht mehr „*WinMain(..)*“, sondern wieder:

```
int main(int argc, char* argv[])
```

Die Anwendung kann lediglich für das automatische Programmieren (Erase, Blank Check, Program und Verify) benutzt werden. Die Hex- oder Binärdatei muss als erster und einziger Parameter an das Programm übergeben werden. Die Hauptfunktion „*main(..)*“ bekommt dann die Anzahl der übergebenden Kommandoparameter in der Variablen „*argc*“ vom Betriebssystem übergeben. Da der eigentliche Dateiname „*usb\_programmer.exe*“ ebenfalls mit in der Kommandozeile steht, muss die Variable bei gültigem Aufruf den Wert 2 haben. Über den Array-Zugriff „*argv[1]*“ kann direkt auf den übergebenen String zugegriffen werden, der den Namen der zu übertragenen Datei enthält.

## 5.8 Der eingesetzte USB-Treiber

Für die Kommunikation unter Windows wird der Gerätetreiber „*ATMUSB.SYS*“ (<http://www.technik-forum.info>) eingesetzt. Der Treiber basiert auf dem EZUSB-Treiber von Cypress und wurde für den Atmel AT89C5131 Controller angepasst. Zur Kompilierung ist das Microsoft DDK erforderlich.

Der Treiber hat unter Windows XP fest definierte Schnittstellen für die Initialisierung und Freigabe (Aufräumcode). Wird der Treiber erstmalig geladen (beim Booten oder Einstecken des Geräts), wird beispielsweise immer die Routine „*DriverEntry*“ des Treibers vom Windows-I/O-Manager aufgerufen.

Wird aus der User-Mode-Anwendung „*usb\_programmer.exe*“ eine Anforderung an den Treiber gestellt, konvertiert der I/O-Manager von Windows die Art der Anforderung (Schreiben, Lesen etc.) in einen Funktionscode und ruft die zuständige Dispatch-Routine des Treibers auf. Für jeden Funktionscode, den der Treiber unterstützt, gibt es eine eigene Dispatch-Routine.

Der Kommunikation der Anwendung mit dem ATMUSB-Treiber erfolgt mit Hilfe der drei Funktionen:

CreateFile(..)	Treiber initialisieren (Handle wird zurückgegeben)
DeviceIoControl(..)	Aufrufen einer speziellen Treiberfunktion
CloseHandle(..)	Handle freigeben und Verbindung zur USB-Pipe beenden

Welche Konstanten das User-Mode-Programm für die Kommunikation mit dem Treiber verwenden kann, erkennt man in der Headerdatei „*atmusb.h*“ im Abschnitt „IOCTL Definitions“. Da der USB-Programmer für die eigentliche Kommunikation nur den Bulk-Transfer-Modus verwendet, sind im Quellcode von „*usb.cpp*“ nur die beiden Konstanten „IOCTL\_BULK\_READ“ und „IOCTL\_BULK\_WRITE“ von Bedeutung.

Die Treiber-Kommunikation ist in die folgenden vier Funktionen gekapselt („*usb.cpp*“):

usb_init(..)	Treiber initialisieren (Handle wird zurückgegeben). Der Kontextname lautet für jedes angeschlossene USB-Gerät dieser Gruppe (max. 8 gleichzeitig möglich): \\.\atmusb-0 \\.\atmusb-1 \\.\atmusb-2 ... \\.\atmusb-7 (Ruft „ <i>CreateFile(..)</i> “ auf)
usb_free(..)	Beendet die Verbindung zur USB Pipe mit dem Treiber und gibt das Handle wieder frei. (Ruft „ <i>CloseHandle(..)</i> “ auf)
usb_send_bulk(..)	Sendet Daten zum BULK-Port des Geräts. (Ruft „ <i>DeviceIoControl(..)</i> “ mit dem Parameter „IOCTL_BULK_WRITE“ auf)
usb_get_bulk(..)	Empfängt Daten vom BULK-Port des Geräts. (Ruft „ <i>DeviceIoControl(..)</i> “ mit dem Parameter „IOCTL_BULK_READ“ auf)

## 6. Anmerkungen und Probleme

### 6.1 Alternative Schaltungs- und Softwarevarianten

Neben dem Atmel AT89C5131 Controller gibt es natürlich auch weitere USB-Chips anderer Hersteller, die sich ebenfalls für die Schaltung geeignet hätten. Die folgende Tabelle zeigt einige Beispiele mit Vor- und Nachteilen der jeweiligen Bausteine:

Hersteller:	Typ:	Vor- / Nachteile:
FTDI	FT245BM	<ul style="list-style-type: none"> <li>- USB wird zu einer virtuellen COM-Schnittstelle (RS232) unter Windows → Windows-Anwendung kann das USB-Gerät nahezu wie ein Gerät mit einer seriellen Schnittstelle behandeln (es gibt fertige Treiber).</li> <li>- nur im SMD-Gehäuse erhältlich (schwieriger zu Löten)</li> <li>- USB ist nicht im Controller integriert, sondern ein Zusatzchip ist erforderlich (→ ein IC mehr)</li> <li>- Die USB-Übertragungsgeschwindigkeit ist etwas langsamer (für den USB-Programmer nicht relevant)</li> <li>- Rel. leicht zu beschaffen (z.B. bei Reichelt)</li> </ul>

Cypress	EZ-USB (AN2131)	<ul style="list-style-type: none"> <li>- Controller auf 8051-Basis mit integrierter USB-Schnittstelle</li> <li>- Rel. günstig und leicht zu beschaffen (z.B. bei Reichelt)</li> <li>- Nicht +5V TTL kompatibel (+3,0 - +3,6V)</li> <li>- Nur im SMD-Gehäuse erhältlich (schwieriger zu Löten)</li> <li>- Gute Dokumentation, mehrere Beispiele</li> </ul>
Philips	ISP1160 PDIUSB12	
Texas Instruments	TUSB 2036 TUSB 3210	
National Semi-conductor	USBN9603 USBN9604	

Auch im Bereich der Software rund um den Mikrocontroller AT89C5131 gibt es Alternativen. Atmel stellt ebenfalls einen kostenlosen USB-Treiber für Windows zur Verfügung. Mit der Anwendung „*Flip*“ (FLexible In-system Programmer) ist das Uploaden neuer Programme in den Flash-Speicher des Controllers im Zusammenspiel mit dem Atmel Treiber möglich. Da die von mir getestete Version jedoch noch einige Probleme hervorgerufen hat (Instabilität, falsche USB-Erkennung), habe ich mich bei diesem Projekt für den ATMUSB.SYS Treiber entschieden!

Die folgende Tabelle zeigt einige alternative Compiler zum SDCC:

<b>Keil <math>\mu</math>Vision Software</b>	Dieses kommerzielle Softwarepaket bietet eine komplette IDE und Projektverwaltung für die Firmware-Entwicklung der C51 Controller. Es gibt sowohl einen C-Compiler mit Editor, als auch einen Simulator für komfortables Debuggen. Das System bietet viele nützliche Funktionen und hat in meinen Tests gut funktioniert (bis zur 2k-Demo-Begrenzung). Die Demo-Version des Programms erlaubt das Erstellen von Hex-Dateien bis zu einer Größe von 2 kByte.
<b>PseudoSam Assembler (A51.EXE)</b>	Assembler für C51 Controller
<b>8051 Cross Assembler (ASM51.EXE)</b>	Assembler für 8051 Controller

## 6.2 Einige Anmerkungen zu anfänglichen Problemen

- In der meisten USB-Literatur (auch im Atmel-Datenblatt) werden die IN- und OUT-Ports (Richtung der Endpoints) immer aus Sicht des PC's betrachtet. Um Verwirrung zu vermeiden hier zwei Beispiele:
  - a) Endpoint 4 ist als Bulk-Out definiert: Das Gerät empfängt auf diesem Endpoint Daten vom PC
  - b) Endpoint 5 ist als Bulk-In definiert: Das Gerät sendet mit diesem Endpoint Daten zum PC
- Atmel liefert speziell für den SDCC-Compiler eine Headerdatei „*at89c5131.h*“ mit Defines für Ports und Register des AT89C5131 Controllers mit. Mit der von mir verwendeten SDCC-Version lies sich die Headerdatei zwar problemlos einbinden, die erzeugten Hex-Dateien haben jedoch nicht funktioniert. Nach Anpassung der Header an die typische SDCC-Syntax war das Problem behoben.

- Beim Umgang und Upload einer neuen Firmware mit Atmels „Flip“ ist es wichtig, dass die eingespielte Hex-Datei den Adressen nach sortierte „Hex-Records“ enthält. Falsche Reihenfolge und Überlappungen erzeugen Probleme.
- Beim SDCC: Interrupt Service Routinen (ISRs), die nicht im Hauptmodul (Quelldatei mit der „main(.)“-Funktion) stehen, müssen unbedingt zusätzlich im Hauptmodul deklariert werden, andernfalls werden sie nicht ausgeführt.

## 6.3 Debugging-Möglichkeiten

Die folgenden Tools und Programme helfen bei der USB-Entwicklung und ermöglichen das Testen und Debuggen während der Firmware-Programmierung:

### 1. USBCommandVerifier (von [www.usb.org](http://www.usb.org))

Dieses Tool vom offiziellen USB Implementers Forum bietet dem Entwickler die Möglichkeit, den USB-Enumeration Prozess detailliert zu analysieren. Der Austausch der verschiedenen Descriptoren und USB-Standard-Requests wird automatisch getestet und auf Fehler überprüft (Testbench-Prinzip). Wurden alle Anfragen konform zur Spezifikation von der Firmware des Geräts verarbeitet, ist der Test erfolgreich („Test Suite Passed“) und das Gerät sollte unter Windows erfolgreich erkannt werden.

Um das Programm benutzen zu können, muss zwingend ein USB-Hub (aktiv oder passiv) an den PC angeschlossen werden. An diesen kann dann das zu überprüfende USB-Gerät gesteckt werden. Wird das Testgerät direkt an den PC gestöpselt, funktioniert die Testsuite nicht.

Nach dem Aufruf des Tools, wird zunächst der Original-USB-Gerätetreiber ersetzt („Stack Switch“), anschließend können die verschiedenen Tests ausgewählt und durchgeführt werden.

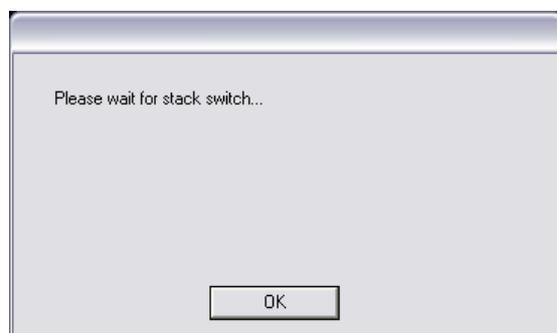


Abbildung 6-1: Zu Beginn des Programms werden die USB-Treiber ausgetauscht

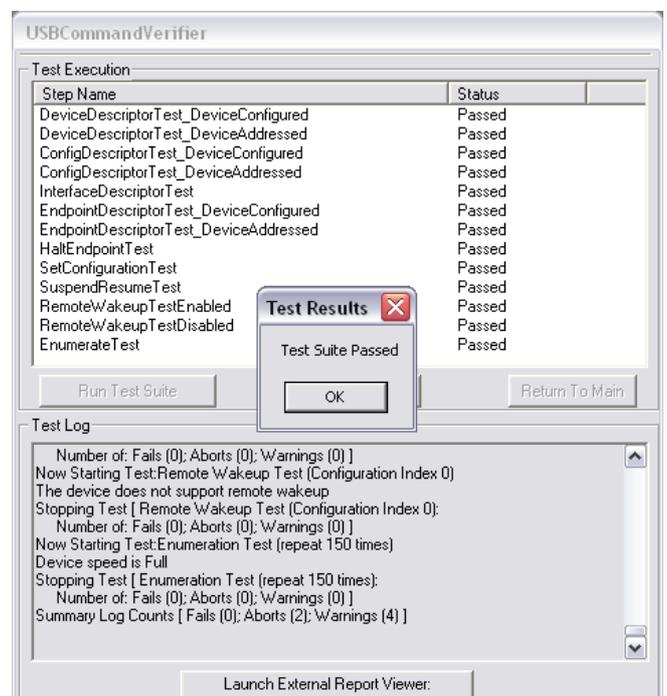


Abbildung 6-2: Alle USB-Standard-Requests werden einzeln getestet. Beim USB-Programmer sind alle Tests erfolgreich durchlaufen!

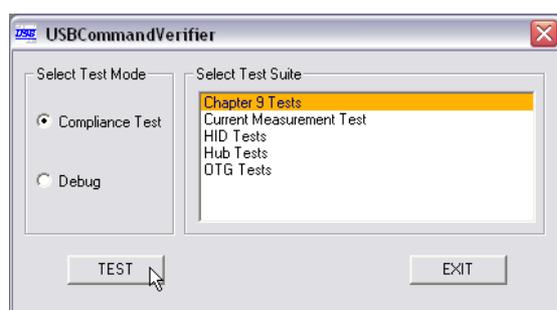


Abbildung 6-3: Chapter 9 Tests überprüft den Paketaustausch während der USB-Enumeration



### 3. USBrowser und USBView

Diese beiden Programme zeigen alle USB-Ports in Windows, sowie die Konfigurationen, Endpoints, Strombedarf, VendorID und weitere Informationen der einzelnen Geräte an. Auch mit diesen Tools lässt sich die korrekte Implementation des USB-Enumeration-Prozesses überprüfen.

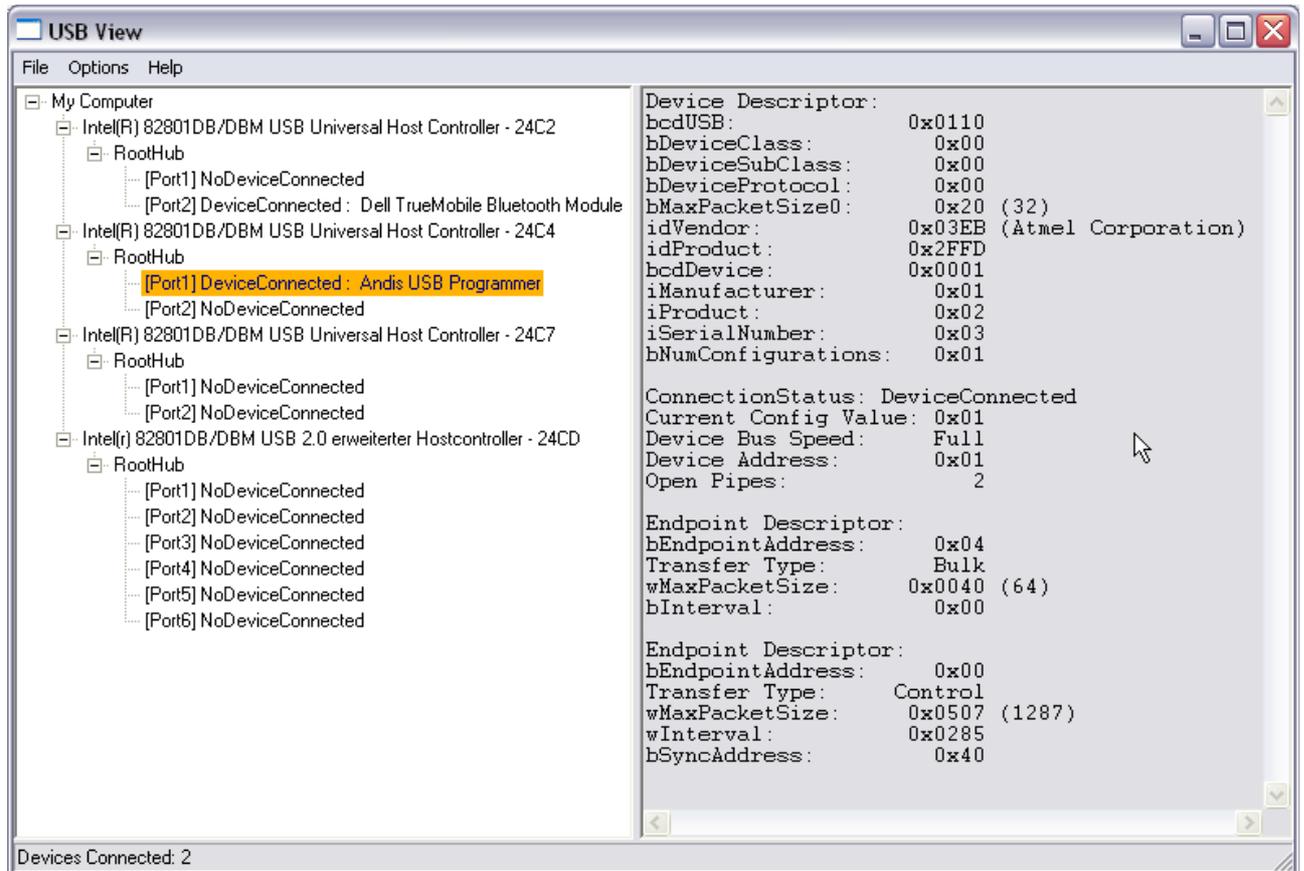


Abbildung 6-6 Das Programm zeigt die wichtigsten Konfigurationsdaten zu den einzelnen Ports

## 7. Anhang

### 7.1 Weiterführende Literatur

- USB 2.0 Handbuch für Entwickler – Jan Axelson
- Messen, Steuern, Regeln mit USB – Burkhard Kainka
- USB Design by Example – John Hyde
- USB in a NutShell (<http://www.beyondlogic.org/>)
- Gerätetreiber unter Windows 2000 – Art Baker, Jerry Lozano

## 7.2 Hilfreiche Internetlinks

Atmel	<a href="http://www.atmel.com/">http://www.atmel.com/</a>
Maxim	<a href="http://www.maxim-ic.com/">http://www.maxim-ic.com/</a>
USB Implementers Forum (Allg. Informationen zu USB)	<a href="http://www.usb.org/">http://www.usb.org/</a>
ATMUSB (USB-Treiber und Technik-Forum)	<a href="http://www.technik-forum.info">http://www.technik-forum.info</a> <a href="http://www.er-tronik.de/">http://www.er-tronik.de/</a>
Keil Software (kommerzielle Entwicklungsumgebung für C51 Controller; Demo arbeitet bis 2k Flash)	<a href="http://www.keil.com/c51/">http://www.keil.com/c51/</a>
SDCC (kostenloser C-Compiler)	<a href="http://sdcc.sourceforge.net/">http://sdcc.sourceforge.net/</a>
The SDCC Open Knowledge Resource (Beispielanwendungen u.v.m.)	<a href="http://sdccokr.dl9sec.de/">http://sdccokr.dl9sec.de/</a>
USB in a NutShell (USB-Informationen)	<a href="http://www.beyondlogic.org/usbnutshell/">http://www.beyondlogic.org/usbnutshell/</a>
SnoopyPro (USB Sniffer für Windows)	<a href="http://sourceforge.net/projects/usbsnoop/">http://sourceforge.net/projects/usbsnoop/</a>

## 7.3 Inhalt der CD-ROM

Die beiliegende CD-ROM hat folgende Verzeichnisstruktur / Inhalt:

<b>Doku</b>	Diese Dokumentation (PDF) und Fotos des USB-Programmers
<b>Quelltexte</b>	Firmware (SDCC) Windows GUI (VISUAL C) Windows Konsole (VISUAL C)
<b>Software</b>	Windows GUI Windows Konsole
<b>Tools</b>	ASMB51 AT89C2051 Beispiel (SDCC) AT89C5131 USB-Bootloader Upload-Tool Atmel USB Keyboard Example ATMUSB-Treiber Quelltext Datenblätter MLASM51 SDCC 2.5.0 SnoopyPro-0.22 USB Bowser USB View usb.org
<b>Treiber</b>	USB-Treiber für Windows XP