

# C5 Die USB-Schnittstelle

In diesem Kapitel soll die Kommunikation über USB möglichst einfach dargestellt werden. Es wird ebenfalls eine kleine USB-Bibliothek für ATMEL®-USB-AVRs® vorgestellt. Die Bibliothek verwendet keine Standardklassen. Sie arbeitet auf der PC-Seite mit der freien „libusb“. Die Software und weitere Erklärungen findet man unter [www.weigu.lu/usb](http://www.weigu.lu/usb).

## Kurze Einführung zu USB

### USB-Transfer

Die USB-Übertragung zwischen PC (*host*) und Gerät (*device, function*) besteht aus mehreren Protokollschichten. Die gesamte Übertragung (Kommunikationsanforderung herstellen und ausführen) wird als **USB-Transfer** bezeichnet. Es existieren **vier** unterschiedliche **Transfertypen**<sup>1</sup>:

- **Control-Transfer**:  
zur Identifikation und Steuerung des Gerätes und für herstellerspezifische Anforderungen (muss von jedem USB-Gerät unterstützt werden und arbeitet immer mit Endpunkt 0 (siehe weiter unten))
- **Bulk-Transfer**:  
zur Übertragung großer Datenmengen ohne garantierte Geschwindigkeit (Bsp.: Laufwerk)
- **Interrupt-Transfer**:  
mit garantierter Bandbreite für geringes Datenvolumen (Bsp.: Tastatur)
- **Isochron-Transfer**:  
mit garantierter Bandbreite aber ohne Fehlerkorrektur (Bsp.: Streaming von Audiodaten)

Ein Transfer besteht aus einer oder mehreren **Transaktionen** (Übergabe eines Dienstes an einen Endpunkt). Es gibt drei Arten von Transaktionen<sup>2</sup>:

- **SETUP (Control)-Transaktion**: bidirektionale Nachrichten
- **IN-Transaktion**: Daten vom Gerät zum PC
- **OUT-Transaktion**: Daten vom PC zum Gerät

Jede Transaktion muss ohne Unterbrechung abgeschlossen werden. Jede Transaktion besteht aus drei Paketen (Phasen)<sup>3</sup>:

---

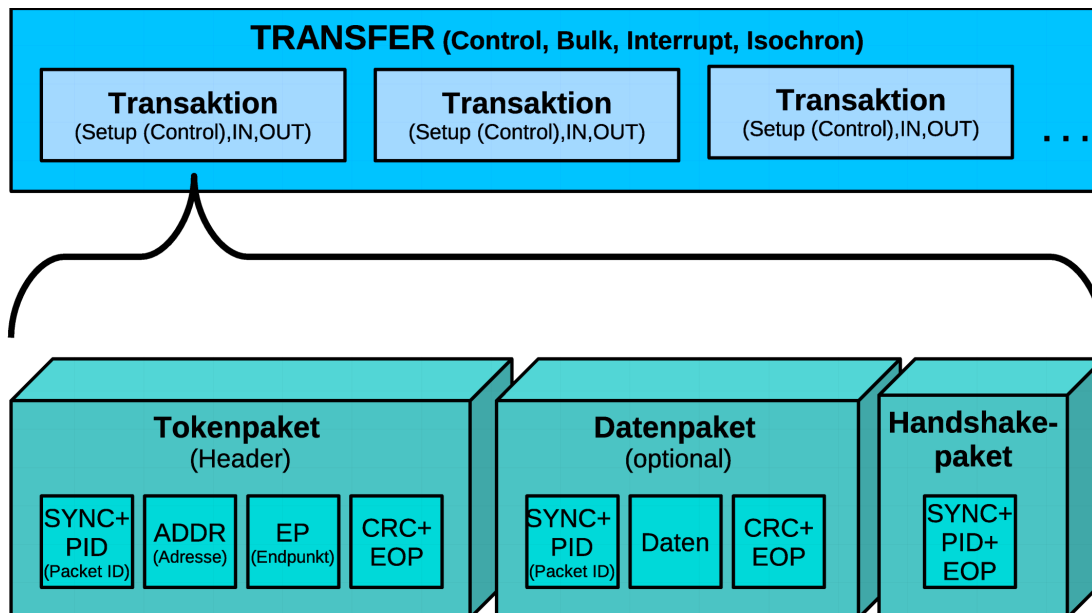
1 Control-Transfers nutzen bidirektionale Nachrichten-Kanäle (Pipes). Die anderen Transfers nutzen unidirektionale Datenstrom-Kanäle.

2 Zur Zeitsteuerung wird auch ein Start-Of-Frame-Transfer bzw. Transaktion durchgeführt. Ebenfalls existiert also ein SOF-Token. Es überträgt eine Zeitreferenz (1ms oder 125µs).

3 Es existiert auch eine vierte "Special"-Phase. Auf sie soll hier nicht eingegangen werden.

- **TOKEN**-Paket:  
gibt an um welchen Transaktionstyp es sich handelt (SETUP, IN, OUT)
- (optionale) **DATA**-Paket:  
enthält Daten oder Statusinformationen (DATA0-2, MDATA)  
Speziell: Beim erfolgreichen Abschließen eines Control-Transfers wird als Status-Information (siehe Handshake) ein Datenpaket ohne Daten (**ZLP**, **Z**ero **L**ength **P**acket) anstelle einer Bestätigung (ACK) versendet.
- **STATUS** (Handshake)-Paket:  
Feedback zur Kommunikation: "Erfolg", "bin beschäftigt (warten)" und "nicht unterstützt" (ACK, NAK, STALL (NYET)).

Die Paketzusammenstellung wird von der Hardware bewerkstelligt. Auf sie soll nicht weiter eingegangen werden.



## Endpunkte

Der Datenaustausch passiert zwischen dem PC und einem spezifischen Geräteendpunkt (*endpoint* EP). Ein Endpunkt ist ein Datenspeicher (FIFO<sup>4</sup>, Puffer, Speicherbank) im Gerät, der meist nur aus 8-256 Bytes besteht. Jedes Gerät hat mehrere Endpunkte die über die Endpunktadresse (015) angesprochen werden. Diese Adresse enthält in Bit 7 ebenfalls die Richtung der Datenkommunikation<sup>5</sup>. Der Control-Endpunkt 0 ist in jedem Gerät vorhanden und wird für zum Beispiel für die Enumeration benötigt. Er ist als einziger Endpunkt bidirektional. Bei USB 1.1 hat er einen 8 Byte großen FIFO-Speicher. Die Anzahl und mögliche Größe der anderen Endpunkte variiert. Jede Transaktion ist an eine Geräte und eine Endpunktadresse gebunden.

4 FIFO (*First In First Out*): Speicherbank, welche zuerst abgelegte Daten beim Lesen auch als erstes wieder ausgibt, im Gegensatz zum LIFO (*Last In First Out*), der zum Beispiel beim AVR Stapel verwendet wird.

5 Außer dem Control-Endpunkt, da dieser bidirektional; Bit 7 = 0: OUT-Endpunkt; Bit 7 = 1: IN-Endpunkt



- 01 Klasse
- 10 Vendor (herstellerspezifische Anfrage)
- 11 Reserviert

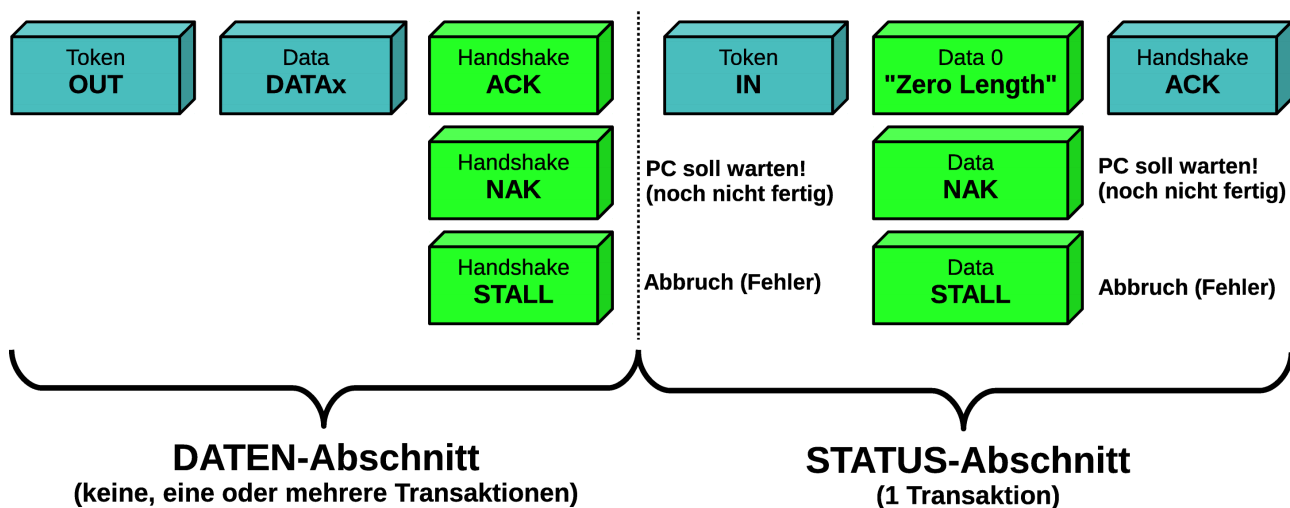
**REC** Recipient REC1, REC0

- 00 Device
- 01 Interface
- 10 Endpoint
- 11 otherReserviert

## Der DATEN- und der STATUS-Abschnitt (Handshake)

### Der schreibende Control Transfer (Data OUT)

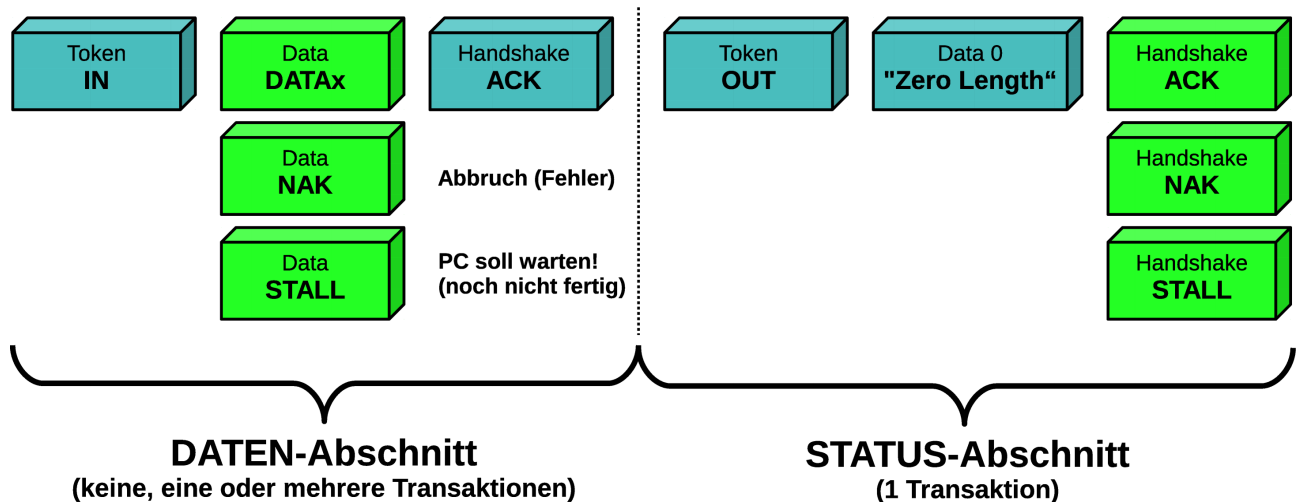
Bei diesem Transfer teilt der PC dem Gerät im Setup-Abschnitt mit, dass er Daten zum Gerät senden will. Bei der Datenphase gibt es nach dem Empfang der Daten dann drei Möglichkeiten. Das Gerät kann den Empfang bestätigen (ACK), den PC auffordern zu warten (NAK) oder Abbrechen (STALL). Tritt beim Token oder Datenpaket ein Fehler auf, so wird das Paket ignoriert. Der DATEN-Abschnitt kann aus mehreren Transaktionen bestehen!



War der Transfer erfolgreich, so sendet das Gerät in der Handshake-Phase ein "Zero Length"-Paket an den PC zurück. Der PC antwortet mit ACK. Trat ein Fehler beim Endpunkt auf, so sendet das Gerät einen STALL. Ist es noch beschäftigt, so sendet es ein NAK. Tritt sonst ein Fehler auf, so wird das Paket ignoriert.

### Der lesende Control Transfer (Data IN)

Bei diesem Transfer teilt der PC dem Gerät im SETUP-Abschnitt mit, dass er Daten vom Gerät lesen möchte. Hier gibt es ebenfalls drei Möglichkeiten. Das Gerät kann den IN Token annehmen und die Daten bereitstellen. Der PC antwortet mit ACK. Ist das Gerät noch beschäftigt, so sendet es ein NAK. Trat ein Fehler beim Endpunkt auf, so sendet es einen STALL. Tritt sonst ein Fehler auf, so wird das Paket ignoriert.



In der Handshake-Phase meldet der PC mit einem "Zero Length"-Paket ob er die Daten erfolgreich erhalten hat. Das Gerät antwortet mit ACK. Fehler oder ein beschäftigtes Gerät geben STALL bzw. NAK zurück. Bei sonstigen Fehlern wird das Paket ignoriert.

## Die Enumeration

Der Vorgang bei dem der PC dem Gerät eine Adresse zuordnet, alle Informationen über das Gerät erfragt, den richtigen Treiber lädt und dann eine Konfiguration auswählt, wird **Enumeration** genannt.

Mittels Standard-Anfragen (*standard request*) wird vom PC die Adresse des Gerätes festgelegt und mehrere sogenannte Deskriptoren (Beschreibungen) erfragt. Jedes Gerät muss mindestens 4 Deskriptoren liefern:

- **Geräte-Deskriptor (*device descriptor*)**
- **Konfigurations-Deskriptor (*configuration descriptor*)**
- **Schnittstellen-Deskriptor (*interface descriptor*)**
- **Endpunkt-Deskriptor (*endpoint descriptor*)**

Es sollen hier nur die aller nötigsten Schritte einer Enumeration vereinfacht beschrieben werden. Wir gehen dabei von nur einer Konfiguration und einem Interface ohne alternative Einstellungen aus.

Nachdem das Gerät an den Bus angeschlossen wurde, ermittelt der PC anhand der Spannungen der beiden Signalleitungen ob ein Low- oder Full-Speed angeschlossen wurde. Der PC resetet dann das Gerät (beide Datenleitungen auf Low) und stellt dabei fest ob es sich um ein Highspeed-Gerät handelt. Nach dem Reset des Gerätes ist dies bereit über Endpunkt 0 auf der Adresse 0 angesprochen zu werden.

Die Standard Anfragen werden dann mittels SETUP-Paketen (siehe Control Transfer) durchgeführt.

Das erste SETUP-Paket (**Get\_Descriptor**) des PC erfragt (Adresse Null, Endpunkt 0) 64 Byte des Geräte-Deskriptor um die maximale Paketgröße von Endpunkt 0 zu ermitteln<sup>7</sup>. Nach 8 Byte (hier befindet sich die `bMaxPacketSize`) bricht der PC ab und führt ein neues Reset des Gerätes aus.

Mit dem zweiten SETUP-Paket (**Set\_Adress**) sendet der PC eine Geräte-Adresse. Diese wird von der Firmware dem Gerät zugewiesen. Das dritte SETUP-Paket (**Get\_Descriptor**) erfragt die 18 Byte des Geräte-Deskriptors. Dann werden mit einem vierten SETUP-Paket (**Get\_Descriptor**) die 9 Byte des Konfigurations-Deskriptors erfragt. Dieses vermittelt die Gesamtlänge des Konfigurations-, Interface- und aller Endpunkt-Deskriptoren. Ein fünftes SETUP-Paket (**Get\_Descriptor**) erfragt all diese Deskriptoren in einer Aktion. Weitere SETUP-Pakete erfragen optionale Deskriptoren (z.B. String-Deskriptoren).

Der PC lädt jetzt anhand von Vendor-ID und Product-ID und der ".INF" Datei den entsprechenden Gerätetreiber.

Ein letztes SETUP-Paket der Emulation (**Set\_Configuration**) bewirkt das Initialisieren und Aktivieren der User-Endpunkte (Endpunkt1-15). Andere, von einer Firmware nicht unterstützte Anfragen des PCs mittels SETUP-Paketen, werden einfach mit STALL abgewiesen.

## Der Geräte-Deskriptor

Es gibt pro Gerät nur einen Geräte-Deskriptor<sup>8</sup>.

Hier der Geräte-Deskriptor mit den Werten für unsere minimale Firmware:

Feldbezeichnung	Byte	Wert	Beschreibung
<code>bLength</code>	1	18 (0x12)	Größe des Deskriptors in Byte
<code>bDescriptorType</code>	1	1	Geräte Deskriptor = 1 (Konstante)
<code>bcdUSB</code>	2	0x0110	USB_Spec1_1
<code>bDeviceClass</code>	1	0xFF	Klassencode (hier anbieterspezifisch = 0xFF)
<code>bDeviceSubClass</code>	1	0xFF	Unterklassencode (hier anbietersp. = 0xFF)
<code>bDeviceProtocoll</code>	1	0xFF	Protokollcode (hier anbieterspezifisch = 0xFF)
<code>bMaxPacketSize</code>	1	8	max. Paketgröße Endpunkt 0 (EP0_FS)
<code>idVendor</code>	2	0x03eb	Atmel Code durch usb.org vergeben
<code>idProduct</code>	2	0x0001	Produkt ID beliebig
<code>bcdDevice</code>	2	0x0001	Release Nummer Gerät
<code>iManufacturer</code>	1	1	Index für String-Deskriptor Hersteller
<code>iProduct</code>	1	2	Index für String-Deskriptor Produkt
<code>iSerialNumber</code>	1	3	Index für String-Deskriptor Seriennummer
<code>bNumConfigurations</code>	1	1	Anzahl möglicher Konfigurationen

<sup>7</sup> Diese kann 8, 16, 32, 64 Byte betragen.

<sup>8</sup> Außer bei Verbundgeräten (*composite device*), die über die Schnittstellen-Deskriptoren statt über den Geräte-Deskriptor beschrieben werden.

## Der Konfigurations-Deskriptor

Ein Gerät kann mehrere Konfigurationen besitzen (bNumConfigurations). Ein Gerät kann zum Beispiel eine Konfiguration für die Stromversorgung per Bus besitzen und eine zweite Konfiguration für eine eigenständige Versorgung. Der Gerätetreiber wählt dann die entsprechende Konfiguration aus. Es ist immer nur eine Konfiguration aktiv.

Im Feld **wTotalLength** wird dem PC die Gesamtzahl der Bytes aller Konfigurations-, Schnittstellen- und Endpunkt-Deskriptoren mitgeteilt. In unserem Fall sind das 9 Byte (1 Konfigurations-Deskriptor) + 9 Byte (1 Schnittstellen-Deskriptor) + 3\*7 Byte (3 Endpunkte neben Endpunkt 0) = 39 Byte.

Feldbezeichnung	Byte	Wert	Beschreibung
bLength	1	9	Größe des Deskriptors in Byte
bDescriptorType	1	2	Konfigurations-Deskriptor = 2 (Konstante)
wTotalLength	2	39 (0x27)	Länge des Konfigurations-Deskriptors und aller untergeordneter Deskriptoren
bNumInterfaces	1	1	Anzahl der Schnittstellen
bConfigurationValue	1	1	Nummer um diese Konfiguration auszuwählen (darf nicht Null sein, sonst geht Gerät in den nicht-konfigurierten Zustand)
iConfiguration	1	0	Index für String-Deskriptor dieser Konfiguration (0 = kein Text)
bmAttributes	1	0x80	D7 = 1 Versorgung durch Bus, D6 = 1 Selbstversorgung, D5 = 1 Remote Wakeup
bMaxPower	1	50	Max. Strombezug vom Bus in 2mA Schritten

## Der Schnittstellen-Deskriptor

Der Schnittstellen-Deskriptor bündelt mehrere Endpunkte zu einer Funktionsgruppe. Es können mehrere Schnittstellen gleichzeitig aktiv sein (Beispiel, Fax-Schnittstelle, Druck-Schnittstelle und Scan-Schnittstelle bei Multifunktionsdrucker). Eine Schnittstelle kann mit der gleichen Schnittstellenummer mehrere alternative Einstellungen beherbergen, welche allerdings nicht gleichzeitig aktiviert werden können. Ein schnelles Umschalten zwischen den alternativen Einstellungen ist möglich.

Feldbezeichnung	Byte	Wert	Beschreibung
bLength	1	9	Größe des Deskriptors in Byte
bDescriptorType	1	4	Schnittstellen-Deskriptor = 4 (Konstante)
bInterfaceNumber	1	0	Anzahl der Schnittstellen
bAlternateSetting	1	0	Nummer um alternative Einstellungen zu wählen
bNumEndpoints	1	3	Anzahl der Endpunkte außer Endpunkt 0
bInterfaceClass	1	0xFF	Klassencode (hier anbieterspezifisch = 0xFF)
bInterfaceSubClass	1	0xFF	Unterklassencode (hier anbietersp. = 0xFF)
bInterfaceProtocol	1	0xFF	Protokollcode (hier anbieterspezifisch = 0xFF)

iInterface	1	0	Index für String-Deskriptor dieser Schnittstelle (0 = kein Text)
------------	---	---	--

## Der Endpunkt-Deskriptor

Die Endpunkt-Deskriptoren beschreiben die Endpunkt (außer Endpunkt 0). Wichtig ist, dass bei der Endpunktadresse auch die Richtung mittels Bit 7 angegeben werden muss.

Hier der Deskriptor für Endpunkt 1 (IN, Bulk, FIFO 8 Byte)

Feldbezeichnung	Byte	Wert	Beschreibung
bLength	1	7	Größe des Deskriptors in Byte
bDescriptorType	1	5	Schnittstellen-Deskriptor = 5 (Konstante)
bEndpointAddress	1	0x81	Bit 7 = 1 (IN), Bit 0-3 Endpunkt-Nummer (andere 0)
bmAttributes	1	2	Transfertyp = Bulk (contr. = 0, iso. = 1, int. = 3)
wMaxPacketSize	2	8	FIFO Größe des Endpunkts in Byte
bInterval	1	0	Polling Interval = 0 (ignoriert für Bulk und Control), 1 für Iso, 1-255 für Interrupt

## Die String-Deskriptoren

Sie sind nicht unbedingt nötig, liefern aber zusätzliche lesbare Informationen. Wird ein String-Deskriptor nicht benötigt, so wird sein Index auf 0 gesetzt.

Strings sind in Unicode (16 Bit) kodiert. Es werden viele unterschiedliche Sprachen unterstützt. Im String-Deskriptor mit dem Index 0 werden die unterstützten Sprachen festgelegt. Hier als Beispiel eine Unterstützung für Englisch und Deutsch:

Feldbezeichnung	Byte	Wert	Beschreibung
bLength	1	6	Größe des Deskriptors in Byte
bDescriptorType	1	3	String-Deskriptor = 3 (Konstante)
wLANGID[0]	2	0x0409	English USA (Standard)
wLANGID[1]	2	0x0407	Deutsch Standard

Als nächster Deskriptor folgt dann der Deskriptor mit Index Eins (hier Hersteller String-Deskriptor)

Feldbezeichnung	Byte	Wert	Beschreibung
bLength	1	18	Größe des Deskriptors in Byte
bDescriptorType	1	3	String-Deskriptor = 3 (Konstante)
bString	16	0x0057,0x0045,.....	"W", "E", "I", "G", "U", " ", "L", "U"

# Firmware AT90USBKEY

## Kurze Beschreibung der Firmware

Der AT90USBKEY ist eine billige kleine Entwicklungsplatine von ATMEL mit einem AT90USB1287-Controller.

Die Firmware besteht aus 2 Dateien (Hauptprogramm, USB-Bibliothek).

Es wird nur eine Konfiguration und ein Interface verwendet. Es stehen neben dem zwingend vorgeschriebenem Endpunkt Null (EP0) noch zwei weitere Endpunkte zur Verfügung. Ein OUT-Endpunkt (EP1) erlaubt dem PC Daten zum Gerät zu senden. Ein IN-Endpunkt (EP2) ermöglicht es dem PC Daten vom Gerät zu lesen.

### Hauptprogramm:

Eine blinkende LED zeigt, dass die Firmware läuft. Mit einem Flag kann das Blinken abgestellt werden.

### USB-Bibliothek

#### **Treiberfunktionen:**

Nach dem Anstecken führt der PC (Host) ein Reset des Gerät (device) aus. Der USB-Teil und die interne PLL des AT90USB1287 werden eingeschaltet und initialisiert. Tritt dann ein End of Reset Interrupt (**EORSTI**) auf, so kann der bei jedem USB-Gerät vorhandene bidirektionelle Control Endpunkt 0 (EP0) initialisiert und aktiviert werden.

Als nächstes sendet der PC ein SETUP-Paket an Endpunkt 0, das durch ein Received SETUP Interrupt (**RXSTPI**) erkannt wird.

#### **Enumeration:**

Standard Requests werden vom PC mittels SETUP-Paketen erfragt.

Das erste SETUP-Paket (**Get\_Descriptor**) des PC erfragt (Adresse Null, Endpunkt 0) 64 Byte des Geräte-Deskriptor um die maximale Paketgröße von Endpunkt 0 zu ermitteln. Nach 8 Byte (hier befindet sich die bMaxPacketSize) bricht der PC ab und führt ein neues Reset des Gerätes aus.

Mit dem folgenden SETUP-Paket (**Set\_Adress**) sendet der PC eine Geräte-Adresse. Diese wird von der Firmware dem Gerät zugewiesen. Das dritte SETUP-Paket erfragt die 18 Byte des Geräte-Deskriptors. Dann werden mit einem vierten SETUP-Paket die 9 Byte des Konfigurations-Deskriptors erfragt. Dieses vermittelt die Gesamtlänge des Konfigurations-, Interface- und aller Endpunkt-Deskriptoren. Ein fünftes SETUP-Paket erfragt all diese Deskriptoren (hier 5) in einer Aktion. Weitere SETUP-Pakete erfragen erfragen die 4 String-Deskriptoren.

Der PC kann jetzt anhand von Vendor-ID und Product-ID und der ".INF" Datei den entsprechenden Gerätetreiber laden.

Ein letztes SETUP-Paket der Emuration (**Set\_Configuration**) bewirkt das Initialisieren und Aktivieren der zwei User-Endpunkte.

Die Firmware antwortet auch noch auf Statusanfragen (**Get\_Status**).<sup>9</sup> Andere, von dieser Firmware nicht unterstützte Anfragen des PC mittels SETUP-Paket werden mit STALL abgewiesen.

## **Anwendungskommunikation:**

Folgende Aktionen können nun von der PC-Software gestartet werden:

- a) Ein- und Ausschalten einzelner Bits (4 LEDs an PD4-PD7 und das Blink-Flag PD0) eines Ports (1 Byte) über Endpunkt 1 (OUT-Aktion (PC ⇒ Gerät)).
- b) Analoge Daten (Analog-Digitalwandler) über den FIFO des Endpunkt 2 einlesen (IN-Aktion (Gerät ⇒ PC)).

## **Zur Kommunikation mit den Endpunkten:**

### **Control Endpunkt 0**

#### **SETUP-Abschnitt:**

Das **Received SETUP** Interrupt (**RXSTPI**) Flag wird gesetzt wenn ein SETUP-Paket eintrifft (SETUP-Abschnitt). In der Firmware wird ein Interrupt ausgelöst. In der Interruptroutine wird ein Unterprogramm zur Behandlung des SETUP-Pakets aufgerufen. In diesem Unterprogramm löscht die Firmware mit **CBI(UEINTX, RXSTPI)**<sup>10</sup> das Interrupt-Flag um das SETUP-Paket zu bestätigen (ACK) und den FIFO-Puffer des Endpunktes zu löschen.

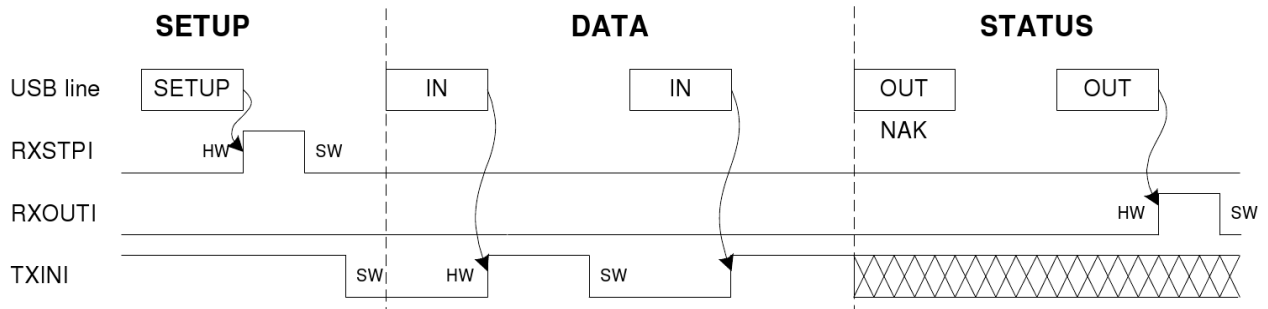
#### **DATA IN und STATUS-Abschnitt:**

Das **TXINI**-Flag (Transmitter Ready Interrupt-Flag) zeigt, dass der FIFO-Puffer frei ist und vom Controller gefüllt werden kann. Nachdem dies geschehen ist wird mit **CBI(UEINTX, TXINI)** das bis zu 8 Byte große Paket (FIFO-Größe EP0) abgeschickt und der FIFO wird wieder gelöscht, so dass er neue Daten aufnehmen kann. Wurden mehr als 8 Byte angefragt, so können mehrere Pakete vor dem Status-Abschnitt gesendet werden. Dazu muss jedes Mal überprüft werden ob der FIFO-Speicher leer ist (**TXINI** = 1).

Das erste OUT-Paket vom PC wird von der Hardware automatisch mit NAK (PC soll warten) beantwortet. Das nächste OUT-Paket setzt das **RXOUTI**-Flag (**Received OUT** Data Interrupt). Das Flag teilt mit, wann das Zero-Length-Paket vom PC angekommen ist. Per Polling wird auf das ZLP gewartet und dann das Flag wieder gelöscht **CBI(UEINTX, RXOUTI)**.

<sup>9</sup> Das Behandeln dieser Anfrage besteht nur um unschöne Fehlermeldungen bei **lsusb -v** unter Linux zu vermeiden (kann bei Bedarf gelöscht werden).

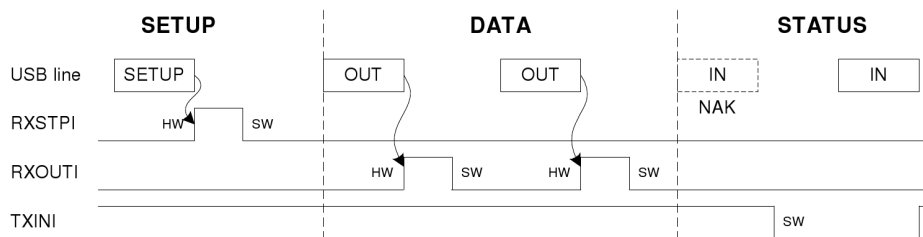
<sup>10</sup> **cbi** (**UEINTX, RXSTPI**) steht für lösche Bit **RXSTPI** im SF-Register **UEINTX**.  
**cbi** **uns** **sbi** (setze Bit) existieren so nur für die untersten 32 SF-Register. Alle USB Register befinden sich im erweiterten Bereich und müssen mittels direkter SRAM Adressierung und Maskierung angesprochen werden.



Quelle: Datenblatt AT90USB128

## DATA OUT und STATUS-Abschnitt:

Soll der PC Daten über den Daten-Abschnitt des Endpunkt 0 an das Gerät senden (was in der Firmware nicht vorkommt), so kann der Datenfluss ebenfalls mit **TXINI** und **RXOUTI** gesteuert werden.

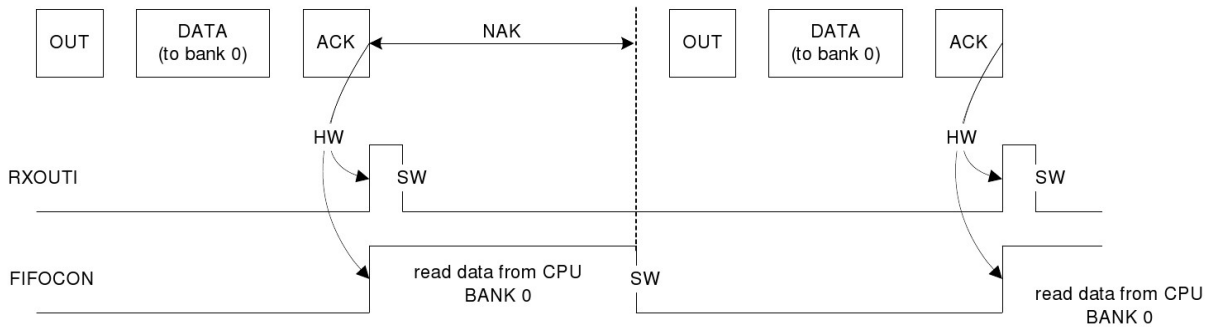


Quelle: Datenblatt AT90USB128

Ein Spezialfall tritt auf wenn keine Daten vorliegen wie bei der **Get\_Address** Anfrage. Es gibt also neben dem SETUP- nur ein STATUS-Abschnitt. Mit **CBI(UEINTX, TXINI)** wird im Statuspaket ein ZLP versendet. Dann wird gewartet bis der Speicher wieder frei ist (**TXINI** = 1, ACK vom PC).

## OUT Endpunkt

Das **Received OUT Data Interrupt (RXOUTI)** Flag wird gesetzt wenn ein OUT-Datenpaket eintrifft. Dieses wird von der Hardware bestätigt. Gleichzeitig setzt die Hardware das **FIFOCON**-Flag (FIFO CONTrOl Bit). In der Firmware wird ein **RXOUT**-Interrupt ausgelöst. In der Interruptroutine wird ein Unterprogramm zur Behandlung des OUT-Pakets aufgerufen. In diesem Unterprogramm löscht die Firmware mit **CBI(UEINTX, RXOUTI)** das Flag um das Interrupt zu bestätigen. Das **RWAL**-Flag im gleichen SF-Register zeigt den Zustand des FIFO. **RWAL** = 1 bedeutet, dass Daten im FIFO vorhanden sind. Sind keine Daten vorhanden, so löscht die Hardware das Flag. Die Firmware überprüft **RWAL** und liest dann die Daten und löscht mit **CBI(UEINTX, FIFOCON)** das **FIFOCON**-Flag um den FIFO-Puffer wieder frei zu geben.



Quelle: Datenblatt AT90USB128

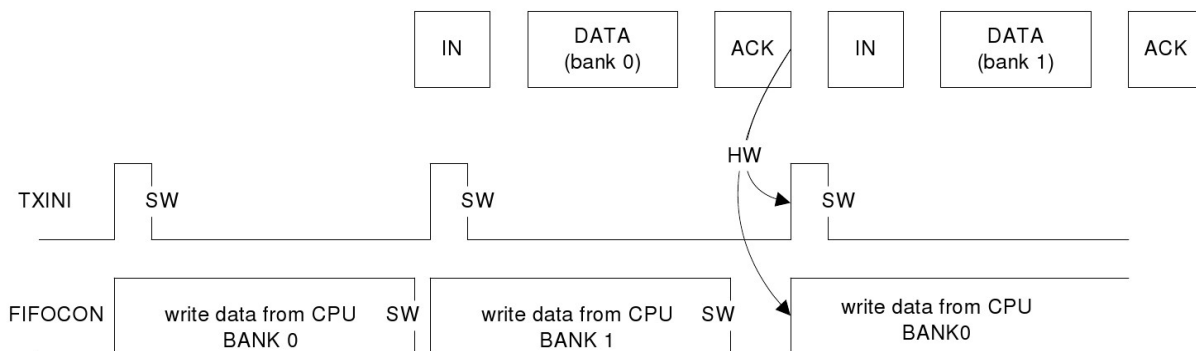
## IN Endpunkt

Benötigt der PC ein Paket, so wird dies über das **NAK IN Received Interrupt (NAKINI)** Flag gemeldet. In der Firmware wird ein Interrupt ausgelöst. In der Interruptroutine wird ein Unterprogramm zur Behandlung der IN-Anfrage aufgerufen. In diesem Unterprogramm löscht die Firmware mit **CBI (UEINTX, NAKINI)** das Interrupt-Flag.

Das **TXINI**-Flag (Transmitter Ready Interrupt-Flag) zeigt, dass der FIFO-Puffer frei ist und vom Controller gefüllt werden kann. Gleichzeitig wird das **FIFOCON**-Flag (FIFO CONTROL Bit) von der Hardware gesetzt.

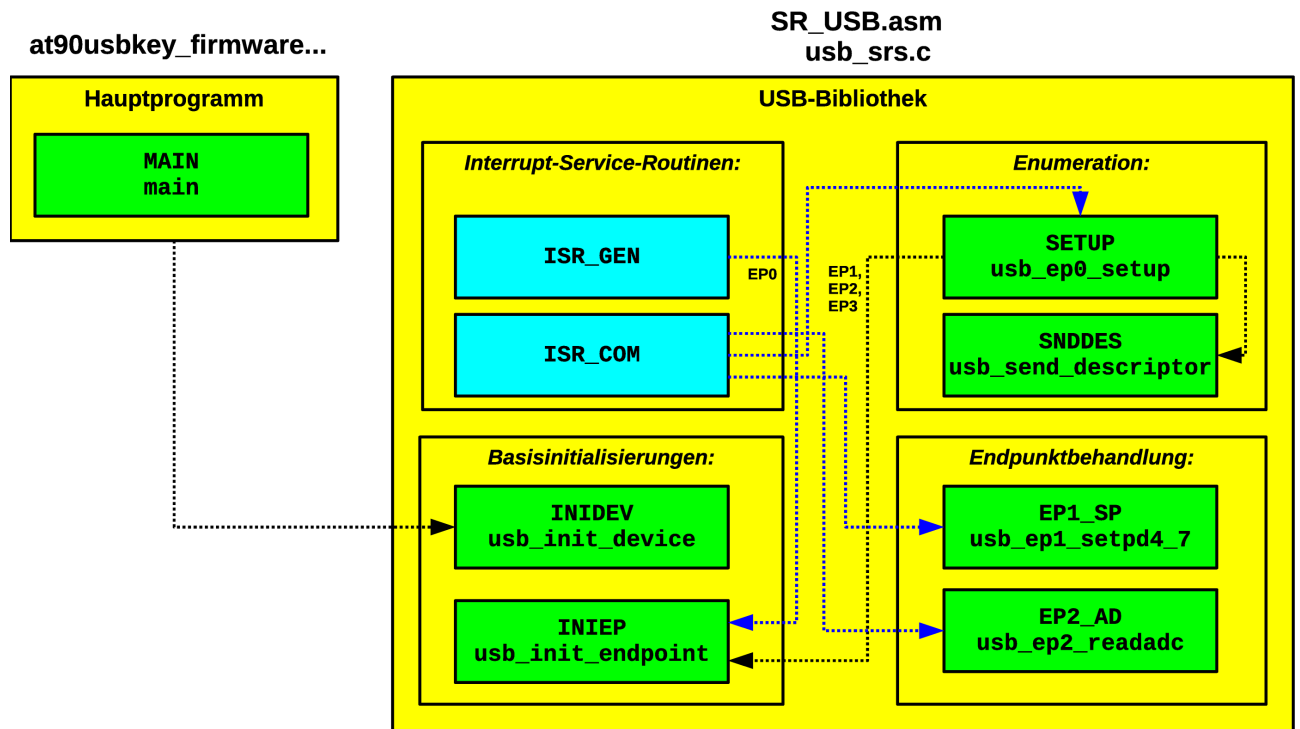
Nachdem dies geschehen ist wird mit **CBI (UEINTX, TXINI)** sofort gelöscht. Der Controller füllt dann den FIFO und erlaubt mit dem Löschen des **FIFOCON**-Flag (**CBI (UEINTX, FIFOCON)**) dem Controller die Daten abzuschicken. Besteht der Endpunkt aus einer Doppelspeicherbank, so wird automatisch auf die nächste Bank umgeschaltet.

(Das **RWAL**-Flag zeigt auch hier den Zustand des FIFO. **RWAL = 1** bedeutet, dass der FIFO voll ist.)



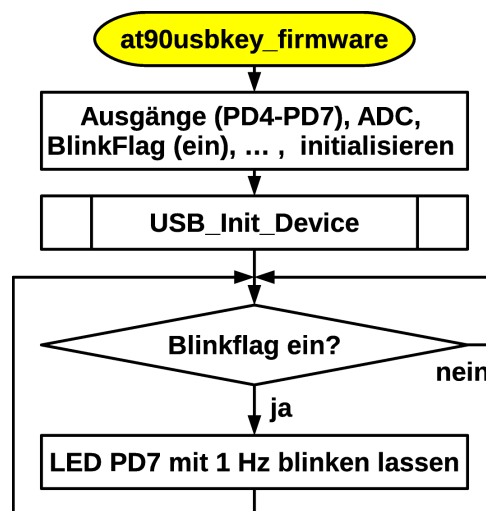
Quelle: Datenblatt AT90USB128

## Übersicht zur Firmware



## Das Hauptprogramm (at90usbkey firmware...)

Im Hauptprogramm zeigt eine LED (**PD4**) durch Blinken mit 1 Hz, dass dieses arbeitet. Ein Flag (**BLINKFLAG**) ermöglicht es das Blinken abzuschalten. Dieses Flag kann später über den Endpunkt 1 mit **PD0** ab- bzw. eingeschaltet werden.



Einzige wichtige Aktion im Hauptprogramm ist der Aufruf des Unterprogramms **INIDEV** (`usb_init_device`), das sich in `sr_usb.asm` (`usb_srs.c`) befindet.

Im Hauptprogramm wird ebenfalls der AD-Wandler initialisiert. Im Assembler muss auch die Vektortabelle im Hauptprogramm initialisiert werden.

## Die USB-Bibliothek (SR USB.asm, usb srs.c)

In der USB-Bibliothek werden die Interrupts abgefangen und behandelt, die Basisinitialisierungen des USB-Teils im ATmega vorgenommen (Treiberfunktionen), die Enumeration wird durchgeführt und die die Endpunkte werden behandelt (Anwenderkommunikation).

### Die Interrupt-Service-Routinen

In der USB-Bibliothek befinden sich **zwei Interrupt-Service-Routinen** (Datenblatt S 252ff).

- Einmal wird der **USB GENeral Interrupt Vektor** behandelt. Dieser Interrupt Vektor kann von 21 verschiedenen Interrupts (General, Device und Host) angesprochen werden. Uns interessiert allerdings nur der **End Of ReSeT** Interrupt (**EORSTI**), welcher das Ende des vom PC auslösten Reset<sup>11</sup> nach dem Anstecken des Gerätes anzeigt (das Gerät hat den Reset Zustand wieder verlassen).
- Die zweite ISR behandelt den **USB Endpoint/Pipe COMMunication Interrupt Vektor (USB COM Interrupt)**. Hier interessieren uns nur drei Interrupts, welche die Endpunkte betreffen<sup>12</sup>. Für **Endpunkt 0** ist das der **Received SETUP** Interrupt (**RXSTPI**), für den bzw. die **OUT-Endpunkte** der **Received OUT Data** Interrupt (**RXOUTI**) und für einen oder mehrere **IN-Endpunkte** der **NAK IN** Received Interrupt (**NAKINI**).

### **USB\_GEN (USB GENeral Interrupt)**

In der Routine für den USB General Interrupt (**ISR\_GEN**) wird überprüft ob sie von einem **End Of ReSeT** Interrupt (**EORSTI**), ausgelöst wurde. War ein anderer Interrupt der Auslöser, so wird die Routine ohne Aktion verlassen.

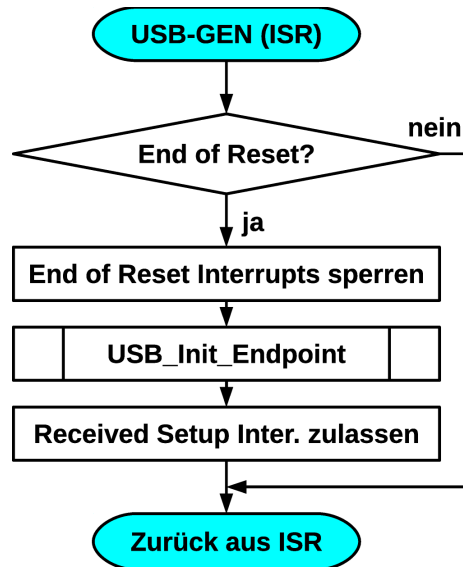
Wurde der Interrupt erkannt, so wird er gesperrt (**CBI**<sup>13</sup> (**UDINT, EORSTI**)), damit er nicht mehr auftreten kann. Der Control **EndPunkt 0 (EP0)** wird über das Unterprogramm **INIEP (usb\_init\_endpoint)** initialisiert. Danach wird der Endpoint 0 **Received SETUP** Interrupt freigeschaltet (**SBI(UEIENX, RXSTPE)**), damit eintreffende SETUP-Pakete erkannt werden können.

11 Der PC zieht dazu beide Datenleitungen gleichzeitig auf Null.

12 Da jeder dieser Interrupts an allen sieben Endpunkten auftreten kann ist es wichtig, vor der Behandlung den richtigen Endpunkt mit dem **ENUM** Register auszuwählen.

13 cbi (**UDINT, EORSTI**) steht für lösche Bit **EORSTI** im SF-Register **UDINT**.

cbi uns sbi (setze Bit) existieren so nur für die untersten 32 SF-Register. Alle USB Register befinden sich im erweiterten Bereich und müssen mittels direkter SRAM Adressierung und Maskierung angesprochen werden.



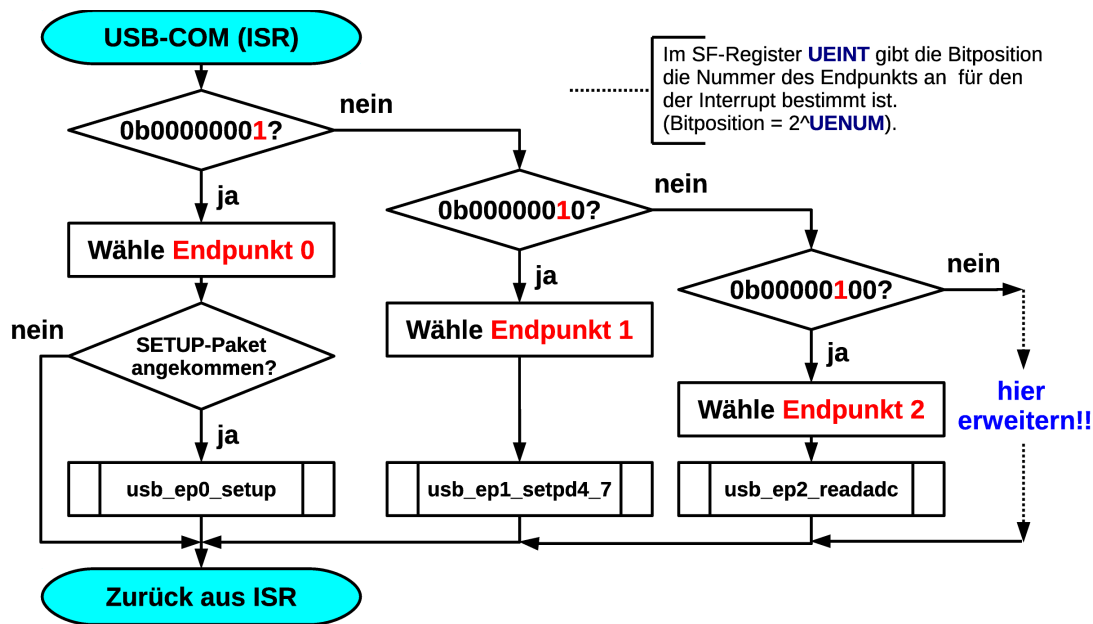
## **USB\_COM (USB Endpoint/Pipe COMMunication Interrupt)**

Die Routine für den USB Endpoint/Pipe COMMunication Interrupt (**ISR\_COM**) reagiert auf die Endpoint Interrupts. Mit dem SF-Register **UEINT** wird erkannt um welchen Endpunkt es sich handelt. Die Bitposition gibt den Endpunkt an. Mit dem SF-Reg **UENUM** muss dann der jeweilige Endpunkt ausgewählt werden. Je nach Endpunkt wird dann eine entsprechende Unterprogramm aufzurufen.

**EP0:** Nur falls ein Received **SETUP** Interrupt (**RXSTPI**) vorliegt wird das Unterprogramm **SETUP** (**usb\_ep0\_setup**) aufgerufen.

**EP1:** Ein Received **OUT** Data Interrupt für Endpunkt 1 bewirkt den Aufruf des Unterprogramms **EP1\_SP** (**usb\_ep1\_setpd4\_7**, im Unterprogramm **SETUP** wurde der **RXOUT** Interrupt freigeschaltet).

**EP2:** Ein **NAK IN** Interrupt für Endpunkt 2 bewirkt den Aufruf des Unterprogramms **EP2\_AD** (**usb\_ep2\_readadc**, im Unterprogramm **SETUP** wurde der **NAKIN** Interrupt freigeschaltet).



## Die Unterprogramme

### Basisinitialisierungen mit INIDEV und INIEP

#### INIDEV (usb\_init\_device)

Hier werden einige Basis-Initialisierungen vorgenommen, damit ein **End Of ReSeT** Interrupt ausgelöst werden kann.

**UHWCON** (USB HardWare CONfiguration) = **0x81**

Per Software wird "Device" d.h. Gerät ausgewählt (PIN UID nicht genutzt!) da unser Gerät keine Host-Funktionen (OTG) übernehmen soll.

Bit 7 (**UIMOD**) = 1 wählt Device-Modus, Bit 0 (**UVREGE** = 1) schaltet den Regelkreis zur Versorgung der D+ und D- Leitung (= pads) mit 3-3,6V ein.

**USBCON** (USB CONfiguration) = **0xB0**

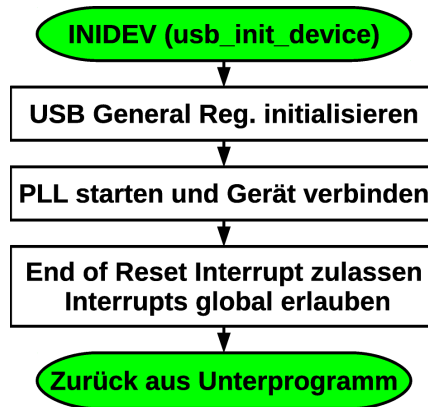
Bit 7 **USBE** = 1 schaltet USB Sender/Empf. und Takt ein. Bit 5 **FRZCLK** bleibt 1 (default Strom sparen). Bit 4 **OTGPADE** = 1 schaltet VBUS-Leitung ein. Muss auch im Device Modus eingeschaltet werden, damit Einstecken und Abstecken vom Bus erkannt wird! Der Takt muss kurzzeitig einschalten werden (**FRZCLK** = 0, **USBCON** = **0x90**), damit ein Transfer möglich wird und somit **EORSTI** auslösen kann (getestet!), kann danach aber gleich wieder abgeschaltet werden um den den Stromverbrauch zu verringern (**USBCON** = **0xB0**).

**UDIEN** = **0x08** (USB Device Interrupt ENable) erlaubt den **End Of ReSeT** Interrupt.

Als nächstes muss die PLL gestartet werden. Die PLL Frequenz wird aus einem 2 MHz Signal durch Multiplikation mit 24 erzeugt. Die 2 MHz werden durch Teilen der Quarzfrequenz erzeugt. Der Vorteiler wird mit den Bits **PLLPO-PLL P2** (3 Bit,  $2^2-2^4$ ) im Register **PLLCSR** (PLL Control and Status Register) initialisiert (0b011 bei 8 MHz-Quarz (:4)) **PLLCSR** = **0x0C**. Nach dem Starten der PLL (**PLLE** = 1, **PLLCSR** = **0x0E**) benötigt die PLL rund 100 ms bevor sie einrastet. Das Flag **PLOCK** im Register **PLLCSR** meldet wenn

das Einrasten erfolgt ist. Es wird einfach mittels Polling abgefragt. Mit **FRZCLK** = 0 (**USBCON** = **0x90**) wird der Takt aktiviert!

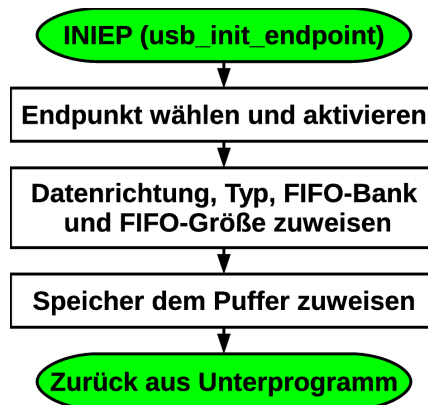
Das Gerät muss mit **UDCON** = 0 (**DETACH** = 0) noch physikalisch verbunden werden (verbindet internen Pull-Up mit der D+ Datenleitung (Full Speed)) und Interrupts mit **sei** global erlaubt werden.



## INIEP (usb\_init\_endpoint)

Das UP **INIEP** dient der Initialisierung und Aktivierung der Endpunkte.

Nach der Auswahl des Endpunkts mit **UENUM** wird dieser aktiviert (**SBI(UECONX, EPEN)**). Über die jeweiligen Bits in **UECFG0X** und **UECFG1X** werden Typ, Richtung, FIFO-Größe und die Art des FIFO-Puffers (einzeln oder doppelt) festgelegt. Mit dem **ALLOC**-Bit wird der Speicher dem Puffer zugewiesen (**SBI(UECFG1X, ALLOC)**)<sup>14</sup>.



	TYPE	DIR	SIZE (FIFO)	BANK
<b>EP0</b>	Control (0)	IN/OUT (0)	8 Byte (0)	1 (0)
<b>EP1</b>	Bulk (2)	OUT (0)	64 Byte (0)	1 (0)
<b>EP2</b>	Bulk (2)	IN (0)	64 Byte (3)	2 (1)

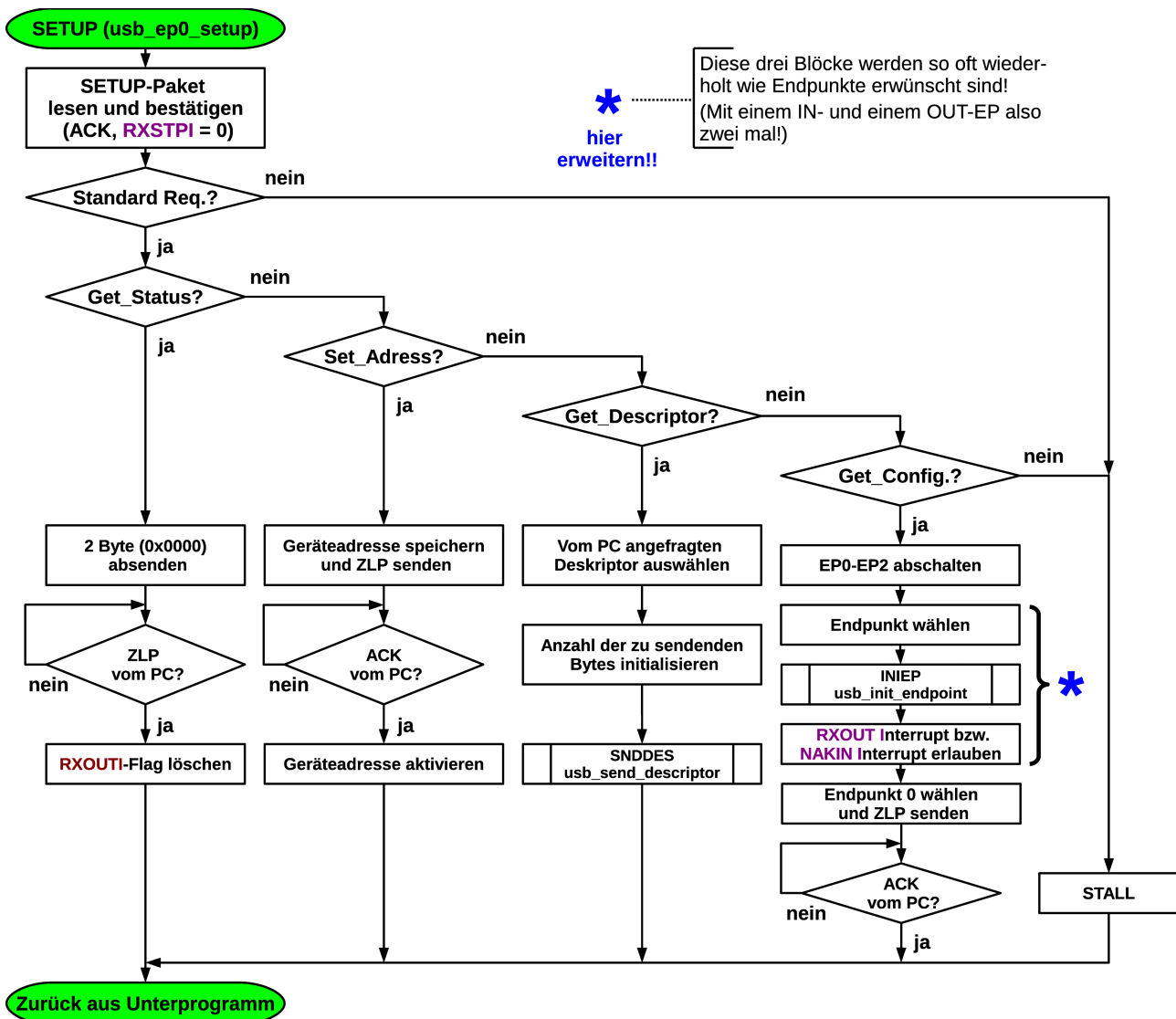
<sup>14</sup> Wie aus dem Flussdiagramm zur Aktivierung eines Endpunkt S269 im Datenblatt ersichtlich kann mit dem Bit **CFGOK** im Register **UECFG0X** dann noch überprüft ob die Zuweisung erfolgreich war. Dies wird hier nicht unterstützt.

## Die Enumeration mit SETUP

Die Enumeration wird durch das Senden von Anfragen (Service Requests) vom PC aus durchgeführt. Der PC sendet SETUP-Pakete mit denen er die Adresse an das Gerät vergibt, Informationen über das Gerät erfragt und die Konfiguration aktiviert. Das Gerät beantwortet die Informationsanfragen mit Deskriptoren. Da jedes Gerät mindestens aus einer Konfiguration und einem Interface bestehen muss werden hier also ein Gerätedeskriptor, ein Konfigurationsdeskriptor, ein Interfacedeskriptor und mehrere Endpunktdeskriptoren erfragt. Zusätzlich können noch Stringdeskriptoren Auskunft über das Gerät oder den Hersteller liefern.

### SETUP (usb\_ep0\_setup)

Dieses Unterprogramm beantwortet SETUP-Pakete an Endpunkt 0 für die Enumeration. Das 8 Byte große SETUP-Paket wird eingelesen, im SRAM abgespeichert und mit ACK bestätigt (**RXSTPI** löschen). Liegt ein Standard Request vor, so wird es sich um einen von den vier in unserer Firmware unterstützten Anfragen handelt. Falls nicht antwortet das Gerät mit STALL.



**Get\_Status:** Dieser Request muss nicht implementiert werden (Zeilen können also gelöscht werden), verhindert aber eine unschöne Fehlermeldung mit „**lsusb -v**“ unter Linux. Es werden zwei leere Byte im FIFO abgelegt (**0x0000**, Flag *remote wakeup* ( $2^1 = 0$ ), wenn das Gerät vom Bus mit Strom versorgt wird (*bus powered*). Hat es eine eigene Versorgung (*self powered*), so muss **0x0001** ins FIFO geschrieben werden. Mit **(CBI(UEINTX, TXINI))** wird das Paket abgeschickt und das FIFO wieder gelöscht. Danach wird auf die Bestätigung des PC gewartet (ZLP).

**Set\_Adress:** Die Adresse wird ermittelt und dem Gerät zugewiesen. Ein Zero-Length Paket (ZLP) meldet den Erfolg. Wurde das ZLP erfolgreich versendet (ACK vom PC), so wird die Adresse aktiviert.

**Get\_Descriptor:** Es wird überprüft, ob es sich um einen Device, Configuration oder String-Deskriptor handelt. Die zu übermittelnden Deskriptoren befinden sich im Flash. Ein Unterprogramm **SNDES (usb\_send\_descriptor)** kümmert sich um das Versenden der Deskriptoren. Dem UP wird in **r18** die Länge des Deskriptors in Byte und in **Z** der Zeiger auf den Deskriptor übergeben.

**Set\_Configuration:** Um die benutzten Endpunkte zu Initialisieren müssen zuerst alle benutzten Endpunkte abgeschaltet werden und ihre Speicherbänke freigegeben werden. Dies passiert in einer Schleife. Mit dem schon bekannten Unterprogramm **INIEP (usb\_init\_endpoint)** werden dann die einzelnen Endpunkte initialisiert.

Nach der Initialisierung eines jeweiligen OUT-Endpunktes muss der **RXOUT<sup>15</sup> Interrupt** für diesen OUT-Endpunkt (**UENUM** = Endpunktnummer!) erlaubt werden. Dadurch kann das Gerät einen neuen COM Interrupt auslösen, wenn Daten vom PC angekommen sind, und dann das zum OUT Endpunkt gehörende Unterprogramm aufrufen, um die Informationen abzuholen.

Der **NAKIN<sup>16</sup> Interrupt** wird für den jeweiligen IN Endpunkt erlaubt (**UENUM** = Endpunktnummer!). Dadurch kann bei einer IN Anfrage des PC ein neuer COM Interrupt ausgelöst werden, der dann das entsprechende Unterprogramm zum IN Endpunkt aufrufen kann, um die angeforderten Informationen zu liefern.

Wurden alle Endpunkte erfolgreich konfiguriert, so wird ein ZLP an den PC gesendet und auf eine Bestätigung (ACK) vom PC gewartet.

## **SNDES (usb\_send\_descriptor)**

In einer Schleife wird der FIFO (Endpunkt 0) mit dem Deskriptor gefüllt. Tritt dabei eine Unterbrechung durch den PC (**RXOUT Interrupt**) auf, so wird abgebrochen. Im Assembler befindet sich die Anzahl der zu schreibenden Bytes in **r18**, der Zeiger auf den Deskriptor in

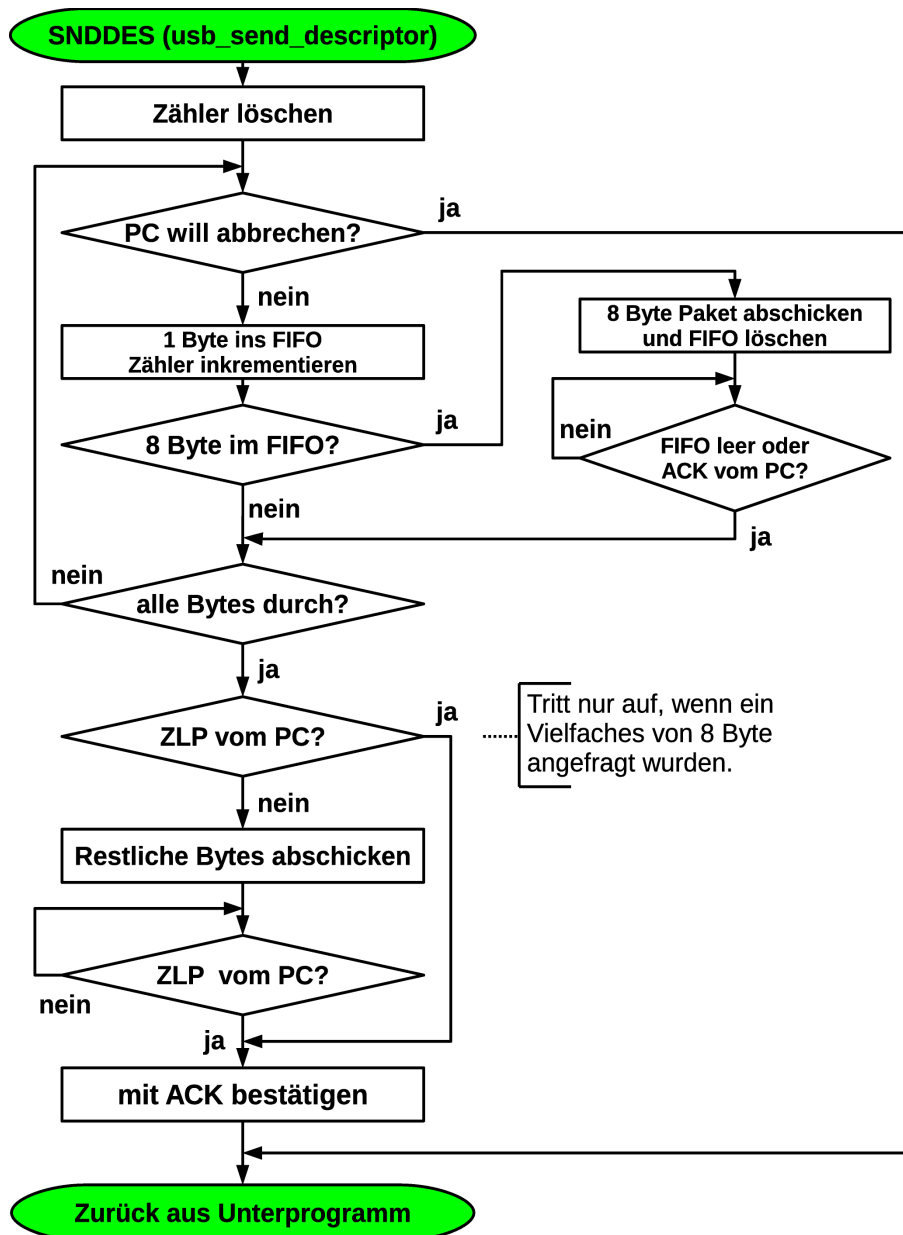
<sup>15</sup> Wird von Hardware gesetzt wenn OUT Daten vorhanden sind.

<sup>16</sup> Wird von Hardware gesetzt wenn IN Anfrage von PC mit NAK beantwortet wurde.

## Z.

Da der FIFO 8 Byte groß ist, wird er mit jeweils 8 Byte gefüllt. Dann wird die Anfrage des PC mit einem ZLP bestätigt und darauf gewartet, dass die Speicherbank wieder frei ist (**TXIN** Interrupt Flag) oder die Daten erfolgreich beim PC angekommen sind (ZLP vom PC, **RXOUT** Interrupt Flag), bevor die nächsten 8 Byte gefüllt werden.

Sind alle angefragten Bytes im FIFO gelandet, so werden die restlichen Bytes abgeschickt, außer es war ein Vielfaches von 8 Byte angefragt worden. Es wird dann das ZLP vom PC gewartet und dieses dann mit einem ACK bestätigt.



## Die Anwendungskommunikation

Die folgenden zwei Unterprogramme dienen der Anwenderkommunikation. UP **EP1\_SP** und **EP2\_AD** werden vom COM Interrupt aus aufgerufen.

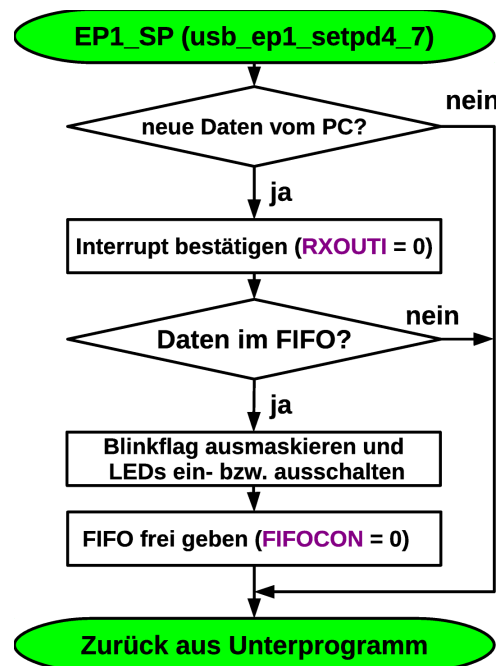
Folgende Aktionen werden dabei ausgelöst:

- EP1\_SP:** Ein- und Ausschalten einzelner Bits (4 LEDs an PD4-PD7 und das Blink-Flag PD0) eines Ports (1 Byte) über Endpunkt 1 (OUT-Aktion (PC  $\Rightarrow$  Gerät)).
- EP2\_AD:** Analoge Daten (Analog-Digitalwandler) über den FIFO des Endpunkt 2 einlesen (IN-Aktion (Gerät  $\Rightarrow$  PC)).

### EP1\_SP (usb\_ep1\_setpd4\_7)

Lese Portbyte von PC und schalte Portbits ein bzw. aus.

Falls der PC Daten für Endpunkt 1 gesendet hat, so wird in der Interruptroutine dieses Unterprogramm aufgerufen. Das Interruptflag (**RXOUTI**) wird gelöscht und mit dem **RWAL**-Flag wird überprüft ob die Daten im FIFO vorliegen und das Gerät die Erlaubnis hat die Daten zu lesen. Das Portbyte wird gelesen und die entsprechenden Aktionen werden durchgeführt (Ein- bzw. Ausschalten des Blinkens und der 4 LEDs). Mit dem Löschen des **FIFO CONTROL** Bit wird gemeldet, dass die Speicherbank jetzt wieder frei ist.



### EP2\_AD (usb\_ep2\_readadc)

Sende analoge Daten an den PC.

Falls der PC Daten am Endpunkt 2 anfragt, so wird in der Interruptroutine dieses Unterprogramm aufgerufen. Als erstes wird eine AD-Wandlung ausgelöst und es wird mittels Polling auf das Ende der Wandlung gewartet. Das Interruptflag (**NAKINI**) wird dann gelöscht und es wird überprüft ob die Speicherbank frei ist. Zwei Byte (10 Bit-

Wandlung) werden ins Fifo geschrieben. Das Abschicken der Daten wird durch Löschen des **FIFO CONTROL** Bit erlaubt.

