Firmware AT90USBKEY

Kurze Beschreibung der Firmware[1]

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 das 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_Deskriptor**) 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

1 Siehe auch „usb_einfuehrung.pdf“

entsprechenden Gerätetreiber laden.
Ein letztes SETUP-Paket der Emuneration (**Set_Configuration**) bewirkt das Initialisieren und Aktivieren der zwei User-Endpunkte.

Die Firmware antwortet auch noch auf Statusanfragen (**Get_Status**).[2] 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 **R**eceived **S**E**T**U**P** **I**nterrupt (**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 Unterprogrsamm löscht die Firmware mit **CBI(UEINTX,RXSTPI)**[3] das Interrupt-Flag um das SETUP-Paket zu bestätigen (ACK) und den FIFO-Puffer des Endpunktes zu löschen.

DATA IN und STATUS-Abschnitt:

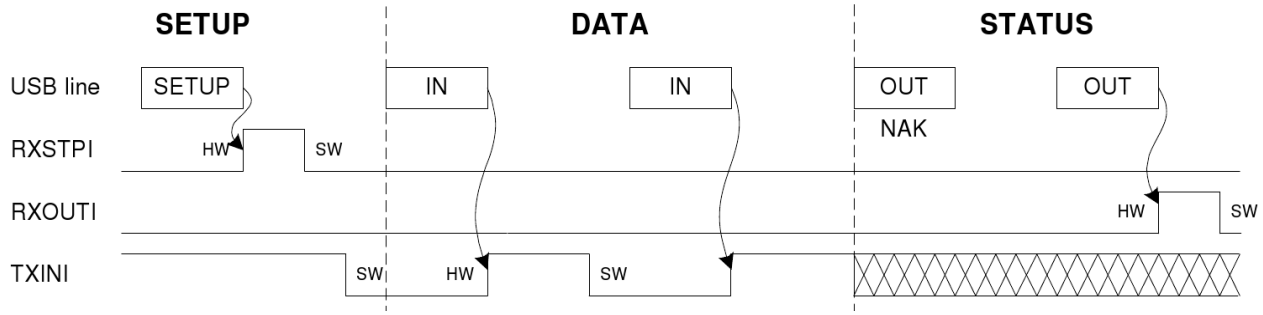
Das **TXINI**-Flag (Transmitter Ready Interrupt-Flag) zeigt, dass der FIFO-Puffer frei ist und vom Controller gefüllt werden kann. Nachdem dies geschehen ist wird mit **CBI(UEINTX,TXINI)** das bis zu 8 Byte große Paket (FIFO-Größe EP0) abgeschickt und der FIFO wird wieder gelöscht, so dass er neue Daten 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 (**R**eceived **OUT** Data **I**nterrupt). Das Flag teilt mit, wann das Zero-Length-Paket vom PC angekommen ist. Per

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

3 **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.

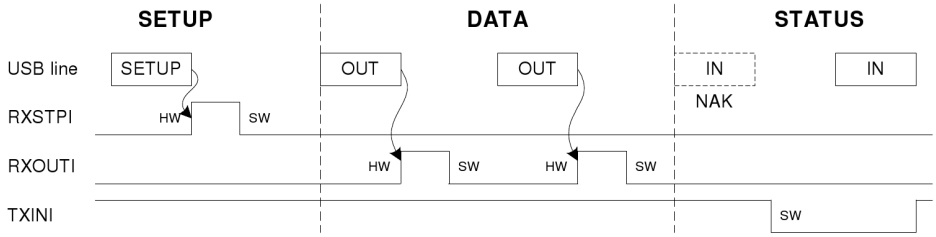
Polling wird auf das ZLP gewartet und dann das Flag wieder gelöscht **CBI(UEINTX,RXOUTI)**.



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.



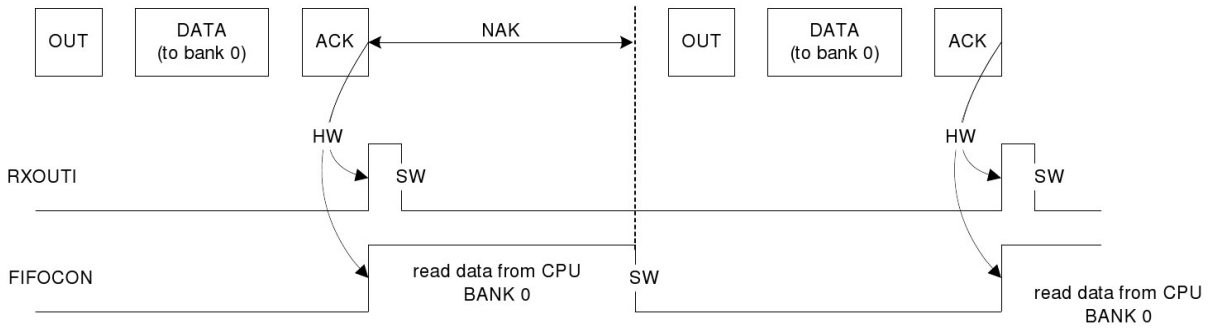
Quelle: Datenblatt AT90USB128

Ein Spezialfall tritt auf wenn keine Daten vorliegen wie bei der **Get_Adress** 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 **R**eceived **OUT** Data **I**nterrupt (**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-I**nterrupt ausgelöst. In der Interruptroutine wird ein Unterprogramm zur Behandlung des OUT-Pakets aufgerufen. In diesem Unterprogrsamm 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

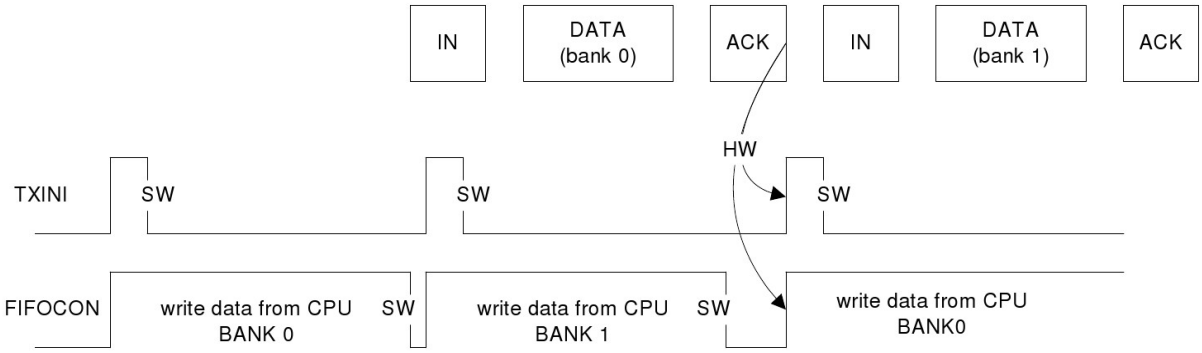
IN Endpunkt

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

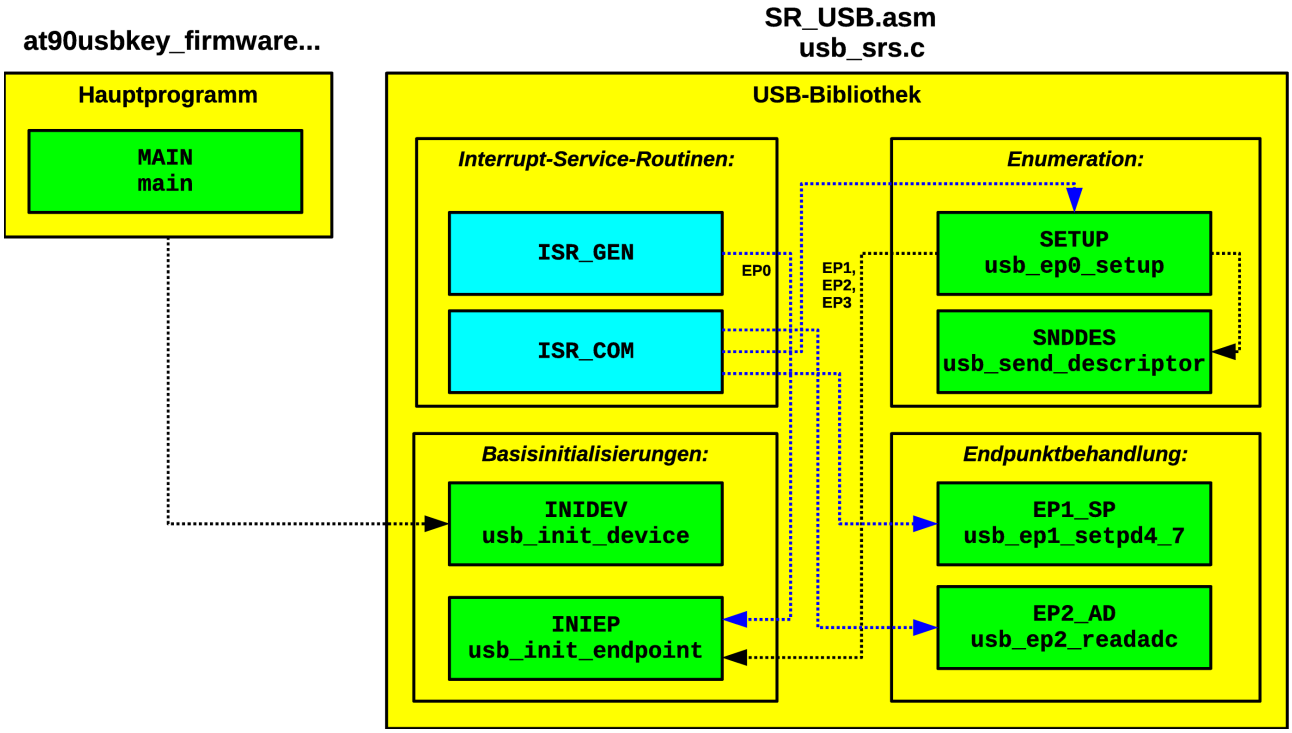
Nachdem dies geschehen ist wird mit **CBI(UEINTX,TXINI)** sofort gelöscht. Der Controller füllt dann den FIFO und erlaubt mit dem Löschen des **FIFOCON**-Flag (**CBI(UEINTX,FIFOCON)**) dem Controller die Daten abzuschicken. Besteht der Endpunkt aus einer Dobbelspeicherbank, so wird automatisch auf die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

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



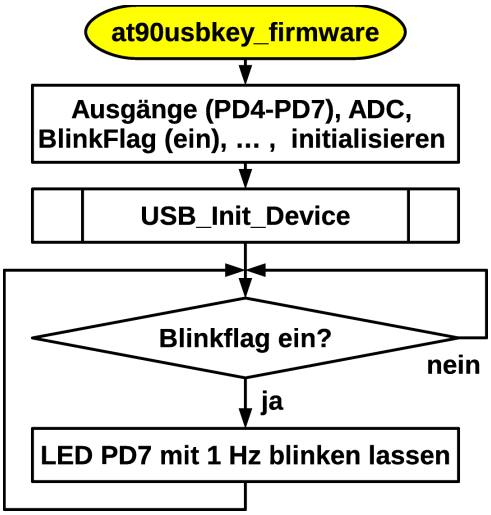
Quelle: Datenblatt AT90USB128

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) angesprungen werden. Uns interessiert allerdings nur der **E**nd **O**f **R**e**S**e**T** **I**nterrupt (**EORSTI**), welcher das Ende des vom PC auslösten Reset[4] 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[5]. Für **Endpunkt 0** ist das der **R**eceived **SETUP** **I**nterrupt (**RXSTPI**), für den bzw. die **OUT-Endpunkte** der **R**eceived **OUT** Data **I**nterrupt

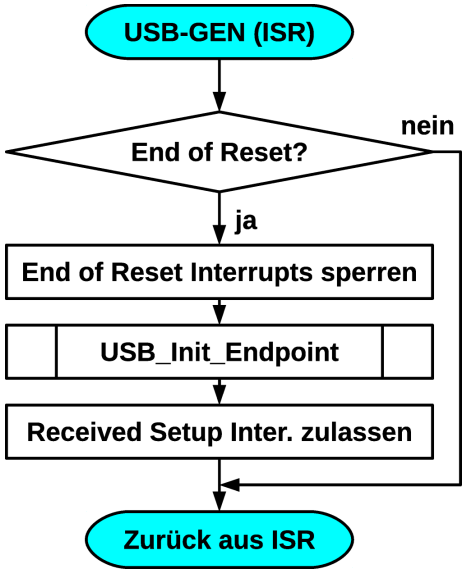
4 Der PC zieht dazu beide Datenleitungen gleichzeitig auf Null.

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

(**RXOUTI**) und für einen oder mehrere **IN-Endpunkte** der **NAK IN** Received **I**nterrupt (**NAKINI**).

USB_GEN (USB GENeral Interrupt)

In der Routine für den USB General Interrupt (**ISR_GEN**) wird überprüft ob sie von einem **E**nd **O**f **R**e**S**e**T** **I**nterrupt (**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**[6] **(UDINT,EORSTI)**), damit er nicht mehr auftreten kann. Der Control **E**nd**P**unkt **0** (**EP0**) wird über das Unterprogramm **INIEP** (**usb_init_endpoint**) initialisiert. Danach wird der Endpoint 0 **R**eceived **S**E**T**U**P** Interrupt freigeschaltet (**SBI(UEIENX,RXSTPE)**), damit eintreffende SETUP-Pakete erkannt werden können.



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.

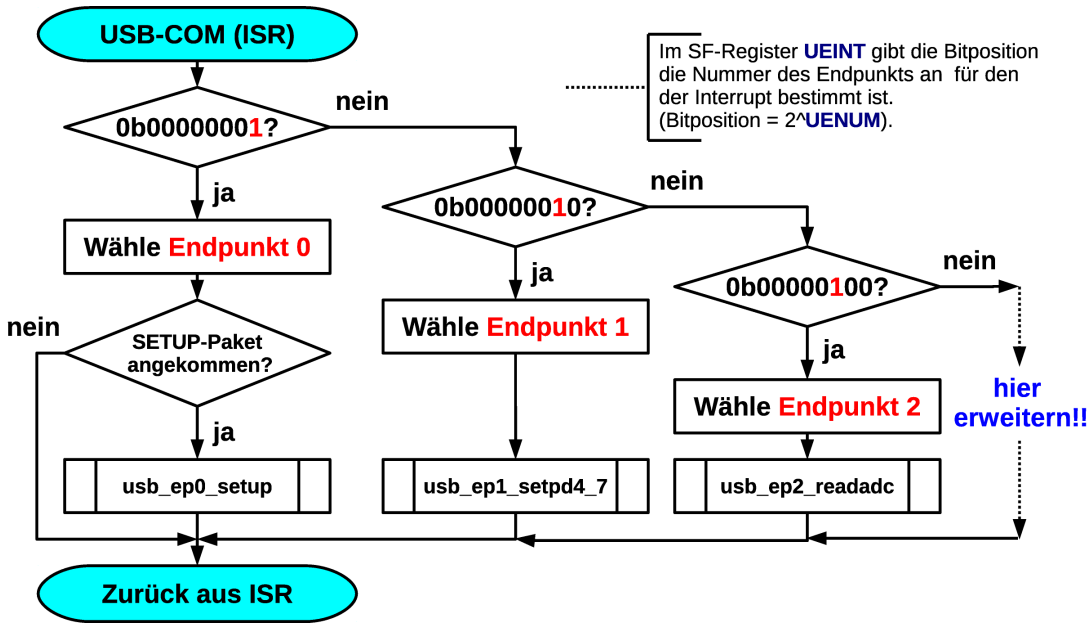
EP0: Nur falls ein **R**eceived **S**E**T**U**P** **I**nterrupt (**RXSTPI**) vorliegt wird das Unterprogramm **SETUP** (**usb_ep0_setup**) aufgerufen.

EP1: Ein **R**eceived **OUT** Data **I**nterrupt für Endpunkt 1 bewirkt den Aufruf des Unterprogramms **EP1_SP** (**usb_ep1_setpd4_7**, im Unterprogramm **SETUP** wurde der **RXOUT I**nterrupt freigeschaltet).

EP2: Ein **NAK IN I**nterrupt für Endpunkt 2 bewirkt den Aufruf des Unterprogramms

6 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.

EP2_AD (**usb_ep2_readadc**, im Unterprogramm **SETUP** wurde der **NAKIN I**nterrupt freigeschaltet).



Die Unterprogramme

Basisinitialisierungen mit INIDEV und INIEP

INIDEV (usb_init_device)

Hier werden einige Basis-Initialisierungen vorgenommen, damit ein **E**nd **O**f **R**e**S**e**T** **I**nterrupt ausgelöst werden kann.

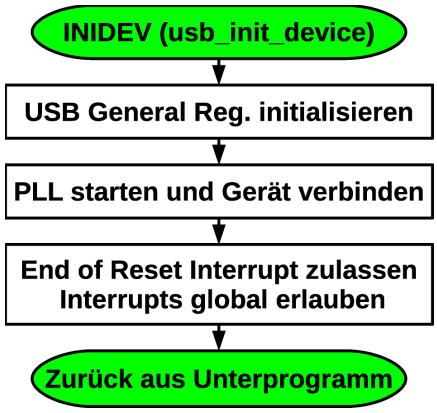
UHWCON (USB HardWare CONfiguration)**= 0x81**
Per Software wird "Device" d.h. Gerät ausgewählt (PIN UID nicht genutzt!) da unser Gerät keine Host-Funktionen (OTG) übernehmen soll.
Bit 7 (**UIMOD**) = 1 wählt Device-Modus, Bit 0 (**UVREGE** = 1)schaltet den Regelkreis zur Versorgung der D+ und D- Leitung (= 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 **E**nd **O**f **R**e**S**e**T** **I**nterrupt.

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

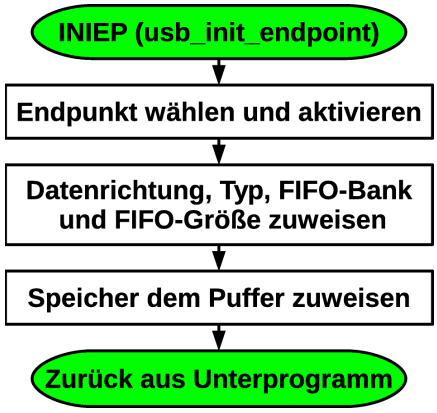
Quarzfrequenz erzeugt. Der Vorteiler wird mit den Bits **PLLP0**-**PLLP2** (3 Bit, 2^2-2^4) im Register **PLLCSR** (PLL Control and Status Register) initialisiert (0b011 bei 8 MHz-Quarz (:4)) **PLLCSR = 0x0C**. Nach dem Starten der PLL (**PLLE** = 1, **PLLCSR = 0x0E**)) benötigt die PLL rund 100 ms bevor 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)**)[7].



7 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.

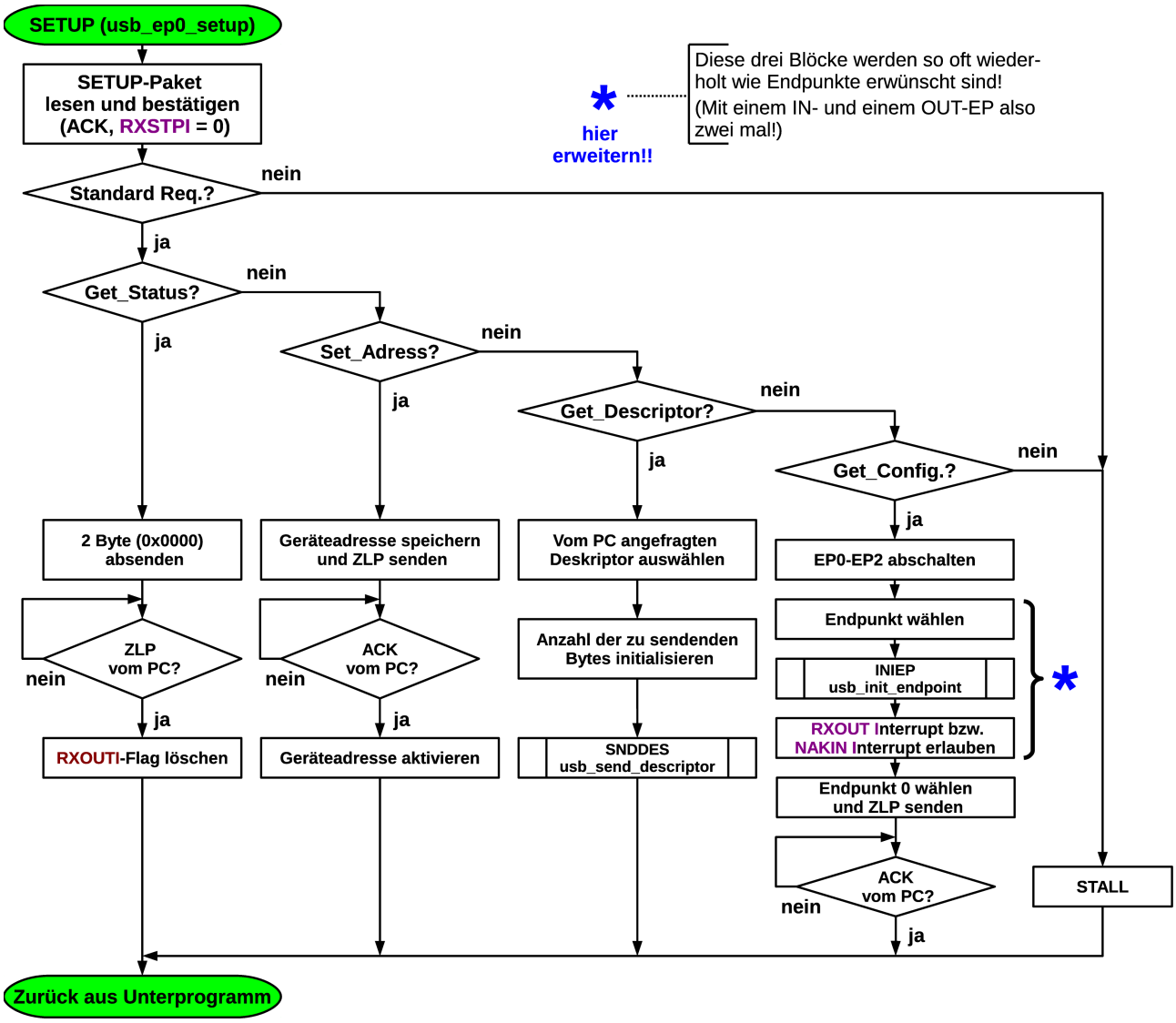
	TYPE	DIR	SIZE (FIFO)	BANK
EP0	Control (0)	IN/OUT (0)	8 Byte (0)	1 (0)
EP1	Bulk (2)	OUT (0)	64 Byte (0)	1 (0)
EP2	Bulk (2)	IN (0)	64 Byte (3)	2 (1)

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 beantworte 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 **`SNDDES`** (**`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.

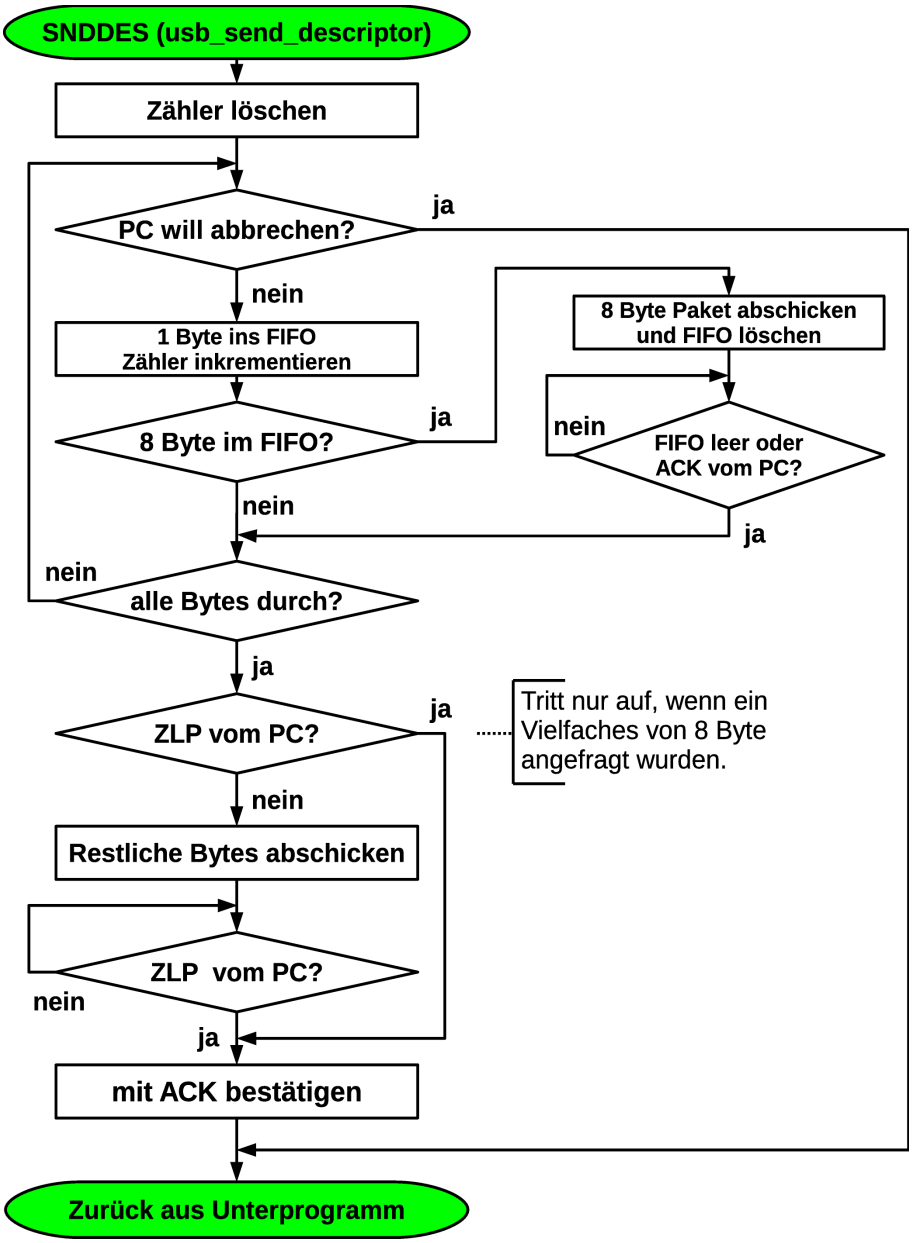
Set_Configuration: Um die benutzten Endpunkte zu Initialisieren müssen zuerst alle benutzten Endpunkte abgeschaltet werden und ihre Speicherbänke freigegeben werden. Dies passiert in einer Schleife. Mit dem schon bekannten Unterprogramm **INIEP** (**usb_init_endpoint**) werden dann die einzelnen Endpunkte initialisiert.
Nach der Initialisierung eines jeweiligen OUT-Endpunktes muss der **RXOUT**[8] **I**nterrupt 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**[9] **I**nterrupt 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.

SNDDES (usb_send_descriptor)

In einer Schleife wird der FIFO (Endpunkt 0) mit dem Deskriptor gefüllt. Tritt dabei eine Unterbrechung durch den PC (**RXOUT I**nterrupt) auf, so wird abgebrochen. Im Assembler befindet sich die Anzahl der zu schreibenden Bytes in **r18**, der Zeiger auf den Deskriptor in **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 I**nterrupt Flag) oder die Daten erfolgreich beim PC angekommen sind (ZLP vom PC, **RXOUT I**nterrupt 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.

8 Wird von Hardware gesetzt wenn OUT Daten vorhanden sind.

9 Wird von Hardware gesetzt wenn IN Anfrage von PC mit NAK beantwortet wurde.



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:

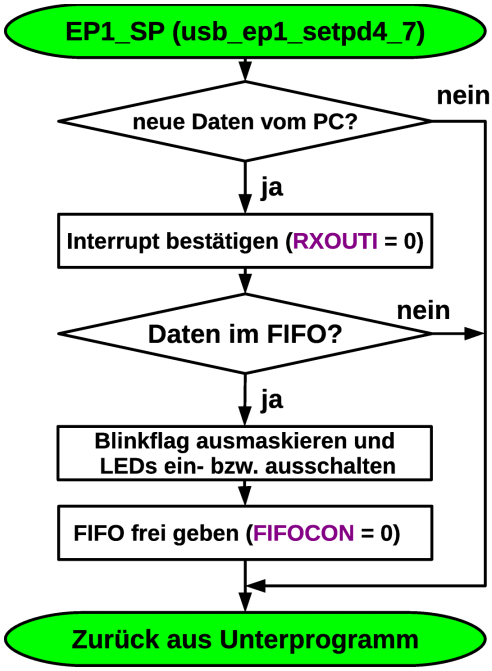
EP1_SP: Ein- und Ausschalten einzelner Bits (4 LEDs an PD4-PD7 und das Blink-Flag PD0) eines Ports (1 Byte) über Endpunkt 1 (OUT-Aktion (PC ⇨ Gerät)).

EP2_AD: Analoge Daten (Analog-Digitalwandler) über den FIFO des Endpunkt 2 einlesen (IN-Aktion (Gerät ⇨ PC)).

EP1_SP (usb_ep1_setpd4_7)

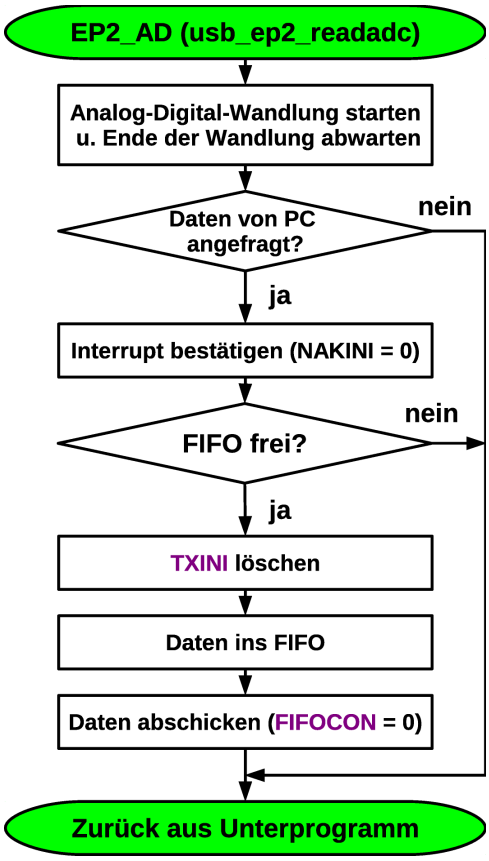
Lese Portbyte von PC und schalte Portbits ein bzw. aus.
Falls der PC Daten für Endpunkt 1 gesendet hat, so wird in der Interruptroutine dieses

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 CON**trol 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 CON**trol Bit erlaubt.


EP2_AD (usb_ep2_readadc)
Analog-Digital-Wandlung starten u. Ende der Wandlung abwarten
Daten von PC angefragt?
nein
ja
Interrupt bestätigen (NAKINI = 0)
FIFO frei?
nein
ja
TXINI löschen
Daten ins FIFO
Daten abschicken (FIFOCON = 0)
Zurück aus Unterprogramm