PROTOKOLL

UART

Gruppe / Klasse	Protokollführer	Unterschrift
5 / 5AHELT	Clemens Uhl	
Übungs-/ Abgabedatum 14.02.2014 21.02.2014	Mitarbeiter Popp Thomas	Unterschrift
Lehrer CRHA	Mitarbeiter Wimmer Michael	Unterschrift
Note	Mitarbeiter Kamil Kaplan	Unterschrift

MESSOBJEKT

FX2
AT89S4051

VERWENDETE GERÄTE

Lfd. Nr.	Art des Geräts	Marke	Typen-Nr.
G_1	Netzgerät	Farnell	E30/2
G_2	Oszilloskop	Tektronik	TDS1002

VERWENDETE PROGRAMME

Keil µVision v4.22.0.0
EzMgr v2.61
Hyperterminal

1 Aufgabenstellung

Die Kernaufgaben dieser Laborübung waren die Schnittstellen USB und RS232.

1.1 USB

- Der Device Deskriptor und Configuration Descriptor des FX2 war mithilfe des Programms EzMgr zu dokumentieren.
- Das serielle Signal der USB-Kommunikation war mit dem Oszillosko zu messen, um den Start of Frame Token anzuzeigen.
- Über den BULK-IN Transfer soll vom FX2 ein String zum PC gesendet werden.

1.2 RS232

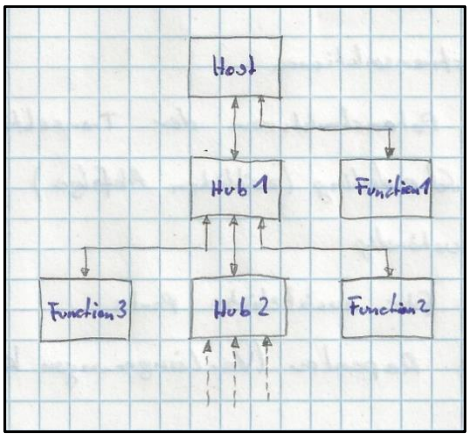
- Der Controller AT89S4051 soll über die RS232-Schnittstelle mit dem Rechner kommunizieren.

2 Grundlegendes

2.1 USB

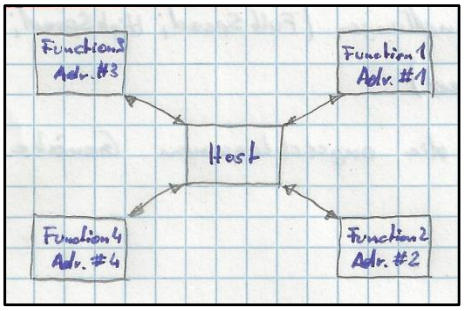
2.2 Struktur von USB

2.2.1 Physikalische Struktur



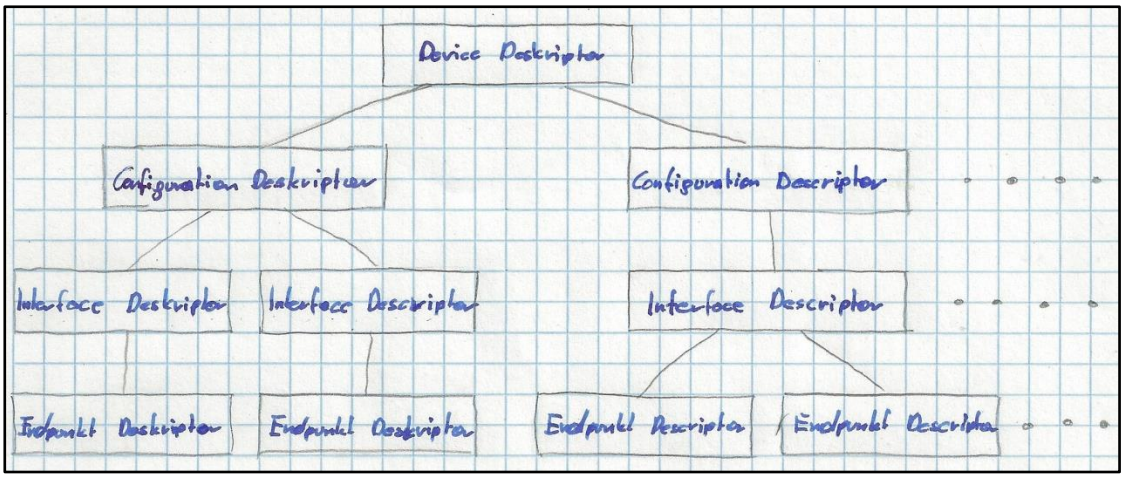
Physikalisch ist USB als Baumstruktur ausgeführt. Maximal können 127 Geräte angeschlossen werden.

2.2.2 Logische Struktur



Die logische Struktur entspricht einem kaskadierten Stern. Es gibt einen Host, der der Angelpunkt aller Kommunikationen ist und den gesamten Busverkehr steuert. Zur zeitlichen Synchronisation aller Geräte am Bus sendet der Host jede ms einen Start of Frame (SOF) Token.

2.3 Deskriptoren



Mit Hilfe der Deskriptoren werden die spezifischen Eigenschaften der USB-Geräte, ihre Konfigurationsmöglichkeiten und die konkreten Endpunkte beschreiben. Die Deskriptoren sind hierarchisch aufgebaut.

2.3.1 Device Descriptor

Jedes Gerät hat einen einzigen Device Deskriptor, der allgemeine Informationen über das Gerät (z.B. Geräteklassen, Typ, Hersteller,…) bereitstellt.

2.3.2 Configuration Descriptor

Ein Gerät kann mehrere Konfigurationen unterstützen, für jede Konfiguration gibt es einen eigenen Configuration Descriptor.

2.3.3 Interface Descriptor

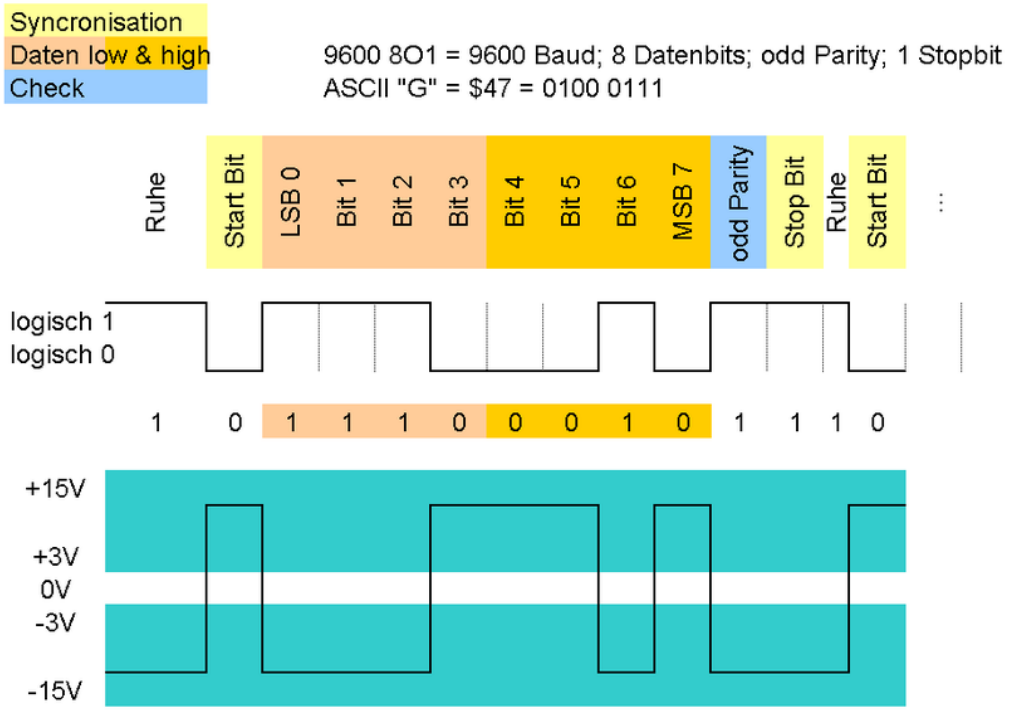
Eine bestimmte Konfiguration kann eine oder mehrere logische Schnittstellen (Interface) aufweisen. Zum Beispiel können USB-DVD-Laufwerke als Massenspeicher oder als Video/Audio-Geräte dienen. Für jede Anwendung gibt es einen anderen Interface Descriptor.

2.3.4 Endpoint Descriptor

Endpunkte sind die tatsächlichen Kommunikationsschnittstellen, die durch die Host-Software auf der Geräteseite angesprochen werden. Es werden sogenannte Communication Pipes zwischen den Endpunkten des Host und des Geräts erstellt. Endpunkt Deskriptoren enthalten Informationen über die Transferart und die unterstützten Transferraten.

2.4 RS232

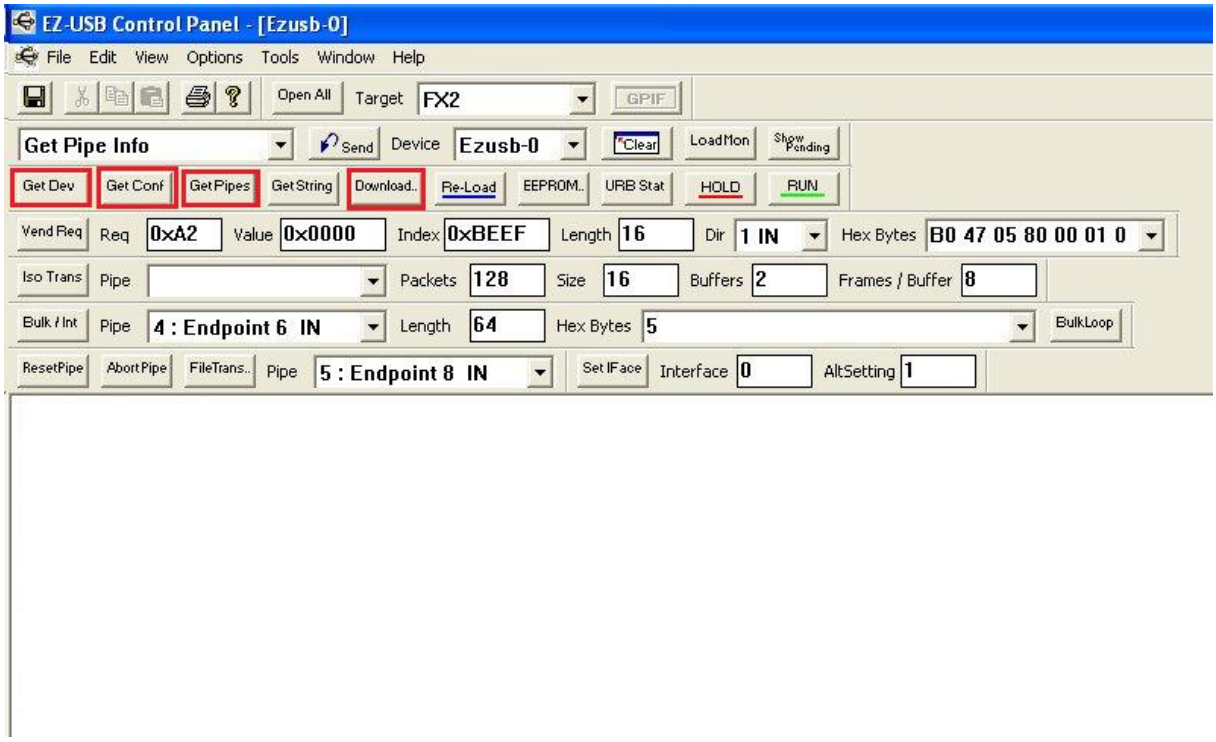
2.4.1 Prinzip



Dieses Bild zeigt das Prinzip der Asynchronen Kommunikation. Der Anfang der Daten wird mit dem Start Bit signalisiert, danach fogen 8 Datenbits. Optional kann das Parity-Bit zur Fehlererkennung hinzugefügt werden. Danach kommt das Stopbit mit variabler Länge (1, 1.5, 2). Die Pegel der RS232-Schnittstelle sind bei einer logischen 1 zwischen -3V und -15V, bei LOW darf der Pegel zwischen +3V und +15V liegen.

3 USB

3.1 Übersicht EzMgr



Der FX2 und der PC wurden mit dem USB-Kabel verbunden. Der FX2 wurde mit dem Programm EzMgr über USB programmiert. Eine Übersicht des Programms zeigt obiges Bild.
Der Befehl „Download“ spielt das Programm in den FX2.
Mit „Get Dev“ kann der Device Descriptor ausgelesen werden.
Mit „Get Conf“ kann der Configuration Descriptor ausgelesen werden.
Mit „Get Pipes“ können die derzeit konfigurierten Communication Pipes angesehen werden.

3.2 Deskriptoren des FX2

3.2.1 Device Descriptor

Mit dem Befehl „Get Dev“ wurde der Geräte Deskriptor ausgelesen. Folgendes Bild ergab sich:

```
Device Descriptor:
bLength:  18
bDescriptorType:  1
bcdUSB:  512
bDeviceClass:  0xff
bDeviceSubClass:  0xff
bDeviceProtocol:  0xff
bMaxPacketSize0:  0x40
idVendor:  0x4b4
idProduct:  0x8613
bcdDevice:  0xa001
iManufacturer:  0x0
iProduct:  0x0
iSerialNumber:  0x0
bNumConfigurations:  0x1
```

Zu sehen sind unter anderem die Geräteklasse sowie Geräteunterklasse, die Vendor ID und die Produkt ID. Letztere zwei sind nötig, da USB kein offener Standard ist und deshalb jedes Gerät eine passende ID benötigt. So werden auch die benötigten Treiber vom Betriebssystem erkannt.

3.2.2 Configuration Descriptor

```
Config Descriptor:
bLength: 0x9
bDescriptorType:  2
wTotalLength:  171 (0xab)
bNumInterfaces:  1
bConfigurationValue:  1
iConfiguration:  0
bmAttributes: 0x80
MaxPower:  50
```

Hier sieht man zum Beispiel den Wert MaxPower, der angibt wie viel Strom das Gerät ziehen darf.

3.2.3 Interface Descriptor

```
Interface Descriptor:
------------------------------
bLength:  0x9
bDescriptorType:  4
bInterfaceNumber:  0
bAlternateSetting:  0
bNumEndpoints:  0
bInterfaceClass:  255 (0xff)
bInterfaceSubClass:  255 (0xff)
bInterfaceProtocol:  255 (0xff)
iInterface:  0
******************************
Interface Descriptor:
------------------------------
bLength:  0x9
bDescriptorType:  4
bInterfaceNumber:  0
bAlternateSetting:  1
bNumEndpoints:  6
bInterfaceClass:  255 (0xff)
bInterfaceSubClass:  255 (0xff)
bInterfaceProtocol:  255 (0xff)
iInterface:  0
```

Hier sind die der Interface Deskriptor abgebildet. Es gibt zwei Configurationsmöglichkeiten, zu sehen unter bAlternateSetting 0 und 1. Erstere Variatne hat keine Endpunkte zur Verfügung, mit der zweiten Konfiguration stehen 6 Endpunkte zur Verfügung.

3.2.4 Endpoint Descriptor

```
Endpoint Descriptor 0                Endpoint Descriptor 3
------------------------------       ------------------------------
bLength:  0x7                        bLength:  0x7
bDescriptorType:  5                  bDescriptorType:  5
bEndpointAddress:  0x1               bEndpointAddress:  0x4
bmAttributes:  0x2                   bmAttributes:  0x2
wMaxPacketSize:  512                 wMaxPacketSize:  512
bInterval:  0                        bInterval:  0
******************************       ******************************
Endpoint Descriptor 1                Endpoint Descriptor 4
------------------------------       ------------------------------
bLength:  0x7                        bLength:  0x7
bDescriptorType:  5                  bDescriptorType:  5
bEndpointAddress:  0x81              bEndpointAddress:  0x86
bmAttributes:  0x2                   bmAttributes:  0x2
wMaxPacketSize:  512                 wMaxPacketSize:  512
bInterval:  0                        bInterval:  0
******************************       ******************************
Endpoint Descriptor 2                Endpoint Descriptor 5
------------------------------       ------------------------------
bLength:  0x7                        bLength:  0x7
bDescriptorType:  5                  bDescriptorType:  5
bEndpointAddress:  0x2               bEndpointAddress:  0x88
bmAttributes:  0x2                   bmAttributes:  0x2
wMaxPacketSize:  512                 wMaxPacketSize:  512
bInterval:  0                        bInterval:  0
******************************       ******************************
```

Hier sind die Endpoint Descriptors zu sehen. Die Werte geben unter anderem die Adresse, den Transfermodus, die Verwendung des Endpunktes an und wie viele Daten über diesen Endpunkt übertragen werden können.

3.3 Kommunikation mit dem FX2

Der FX2 sendet einen Text via BULK-IN-Transfer auf den Endpunkt 6. Der Text wurde mit einem selbst erstellten Programm und mit dem EzMgr angezeigt.

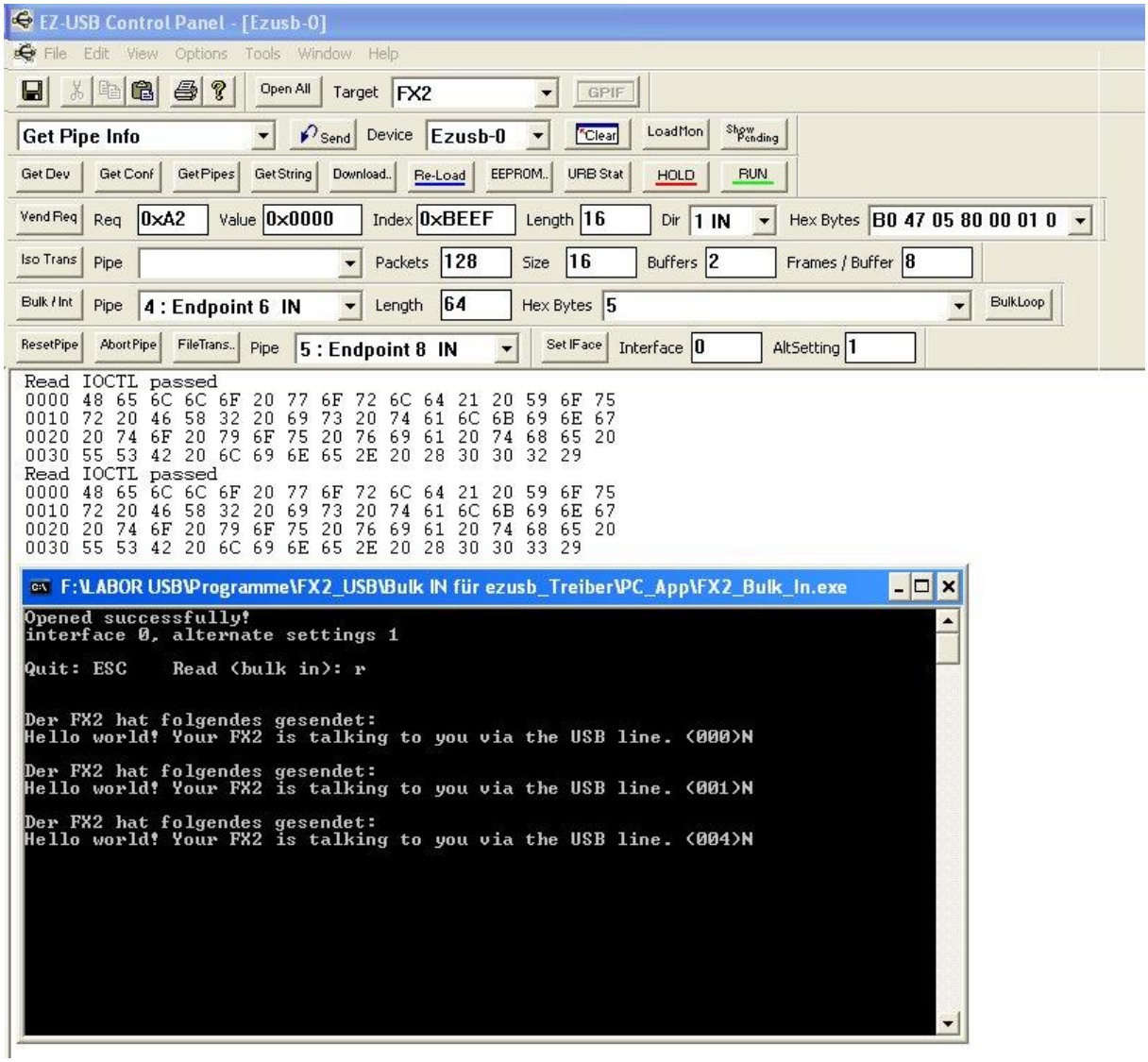
3.3.1 Programm des FX2

```
#define ALLOCATE_EXTERN
#include <fx2.h>
#include <fx2regs.h>
#undef ALLOCATE_EXTERN
#include <intrins.h>


// Read TRM p.15-115 for an explanation on this.
// A single nop is sufficient for default setup but like that we're
on
// the safe side.
#define    NOP        _nop_ ()
#define    SYNCDELAY  NOP; NOP; NOP; NOP
```

```
static void Initialize(void)
{
     CPUCS=0x10;   // 48 MHz, CLKOUT output disabled.

     IFCONFIG=0xc0;  // Internal IFCLK, 48MHz; A,B as normal ports.
     SYNCDELAY;

     REVCTL=0x03;  // See TRM...
     SYNCDELAY;

     EP6CFG=0xe2;  // 1110 0010 (bulk IN, 512 bytes, double-
buffered)
     SYNCDELAY;

     FIFORESET = 0x80;  SYNCDELAY;  // NAK all requests from host.
     FIFORESET = 0x82;  SYNCDELAY;  // Reset individual EP (2,4,6,8)
     FIFORESET = 0x84;  SYNCDELAY;
     FIFORESET = 0x86;  SYNCDELAY;
     FIFORESET = 0x88;  SYNCDELAY;
     FIFORESET = 0x00;  SYNCDELAY;  // Resume normal operation.
}


// This will put some data into the EP6 buffer and make it ready for
// transmission.
static void SetUpBufToTransfer(void)
{
     // Call serial.
     static unsigned char serial=0;

     // First, copy the data into the EP6 buffer.
     xdata unsigned char *dest=EP6FIFOBUF;
     const char *src=
          "Hello! Wimmer is scheisse!!! ";
     unsigned char len=0;
     while(*src)
     {
          *dest++=*src++;
          ++len;
     }
     // Append call serial in decimal:
     *dest++='(';                ++len;
     *dest++='0'+serial/100;     ++len;
     *dest++='0'+serial%100/10;  ++len;
     *dest++='0'+serial%10;      ++len;
     *dest++=')';                ++len;
     ++serial;

     // Arm the endpoint. Be sure to set BCH before BCL because BCL
access
     // actually arms the endpoint.
     SYNCDELAY;  EP6BCH=0;
     SYNCDELAY;  EP6BCL=len;
```

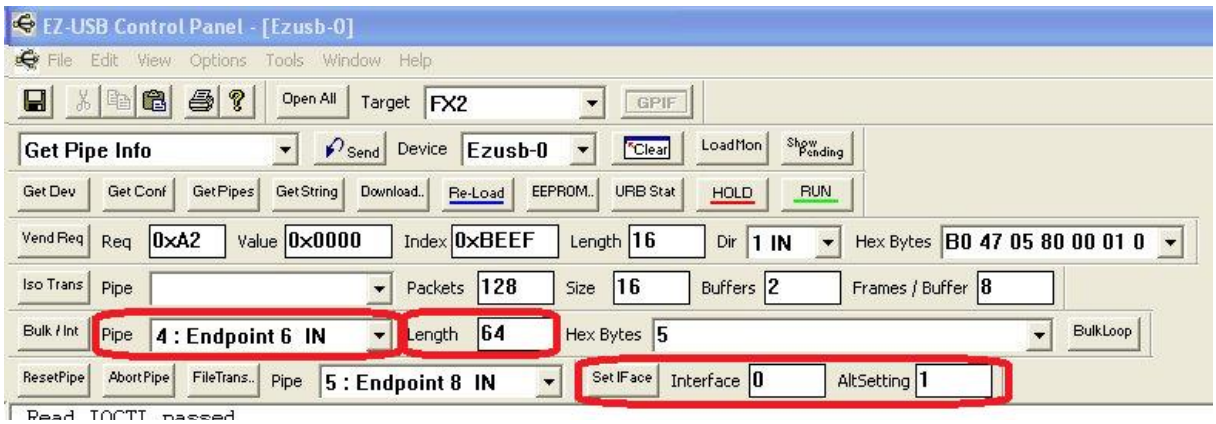
```
	// That's all we gonna do; the data will be transmitted the
next time
	// a bulk read is performed by the host.
}


void main(void)
{
	Initialize();

	for(;;)
	{
		// Wait for the EP6 buffer to become non-full.
		if(!(EP6CS & (1<<3)))
		{  SetUpBufToTransfer();  }
	}
}
```

3.3.2 Angezeigter Text



Durch Drücken der Taste r im Programm wird der Text empfangen. Der FX2 zählt mit, wie oft er die Botschaft sendet. Im Konsolen-Fenster ist zu sehen, dass der

Zähler von 1 auf 4 zählt. Das kommt dadurch zustande, dass die Nachrichten 2 und 3 mit dem EzMgr empfangen worden sind. Dafür sind folgende Einstellungen zu treffen:



Danach können mit einem Klick auf „Bulk/Int“ die Daten empfangen werden. Sie werden jedoch hexadezimal im ASCII-Code angezeigt, was auf der vorigen Seite ersichtlich ist.

3.3.3 Communication Pipes

Wenn das Programm in den FX2 gespielt wird, werden die Communication Pipes konfiguriert. Das kann im EzMgr mit einem Klick auf „Get Pipes“ überprüft werden.

Vor dem Einspielen des Programms zum Senden

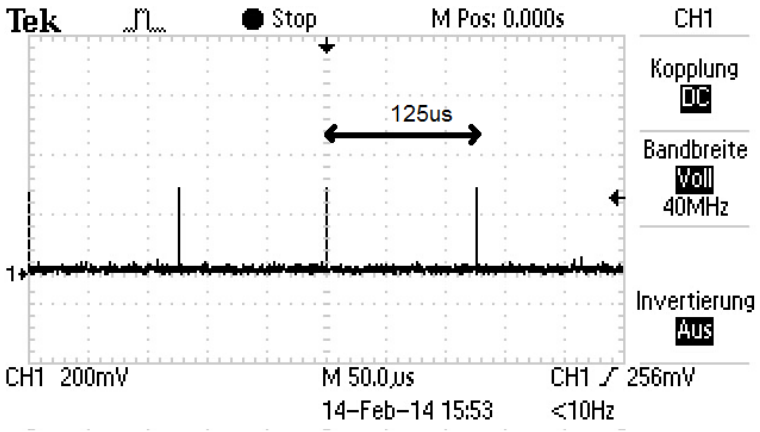
```
Get PipeInfo
Interface Size 16
```

Nach dem Einspielen des Programms zum Senden

```
Get PipeInfo
Interface Size 136
Pipe:  0   Type: BLK Endpoint:  1 OUT  MaxPktSize: 0x200
Pipe:  1   Type: BLK Endpoint:  1 IN   MaxPktSize: 0x200
Pipe:  2   Type: BLK Endpoint:  2 OUT  MaxPktSize: 0x200
Pipe:  3   Type: BLK Endpoint:  4 OUT  MaxPktSize: 0x200
Pipe:  4   Type: BLK Endpoint:  6 IN   MaxPktSize: 0x200
Pipe:  5   Type: BLK Endpoint:  8 IN   MaxPktSize: 0x200
```

Nun sind die Pipes aufgelistet. Ersichtlich sind die Endpunkte sowie die maximalen Daten (hier: 512 Bytes).

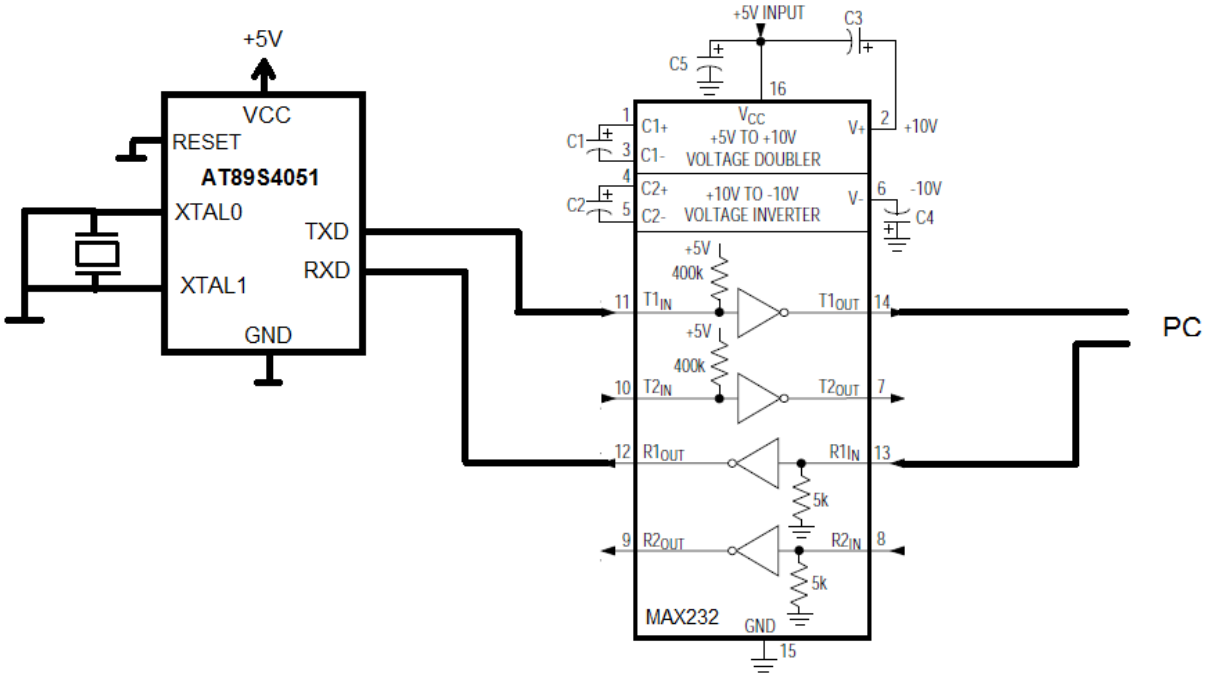
3.3.4 Start of Frame Token



Der Abstand zwischen zwei SOF-Tokens ist 125us.

4 RS232

4.1 Schaltung



Dies ist die Schaltung um den Controller mit der RS232-Schnittstelle des PCs zu verbinden. Der MAX232 ist nötig, um die Spannungspegel an den Standard der RS232 anzupassen. Die Kondensatoren C1 bis C5 sind 1uF groß. Auf Kondensatoren beim Quarz wurde verzichtet, da die Steckbrettkapazität ausreichen war.

4.2 Berechnung der Baudrate

Der Controller verwendet Timer, um die gewünschte Bitdauer und somit Baudrate zu erzeugen. Im Datenblatt ist dazu eine Formel zu finden.

$$TH1 = 256 - \frac{f_{Quarz}}{12} * \frac{2^{SMOD}}{Baudrate * 32}$$

SMOD ist ein Bit im Register PCON, das den Wert 0 oder 1 haben kann und somit den einstellbaren Bereich vergrößert.
In der Laborübung wurde ein Quarz mit 8.867238 MHz und eine Baudrate von 4800 verwendet.

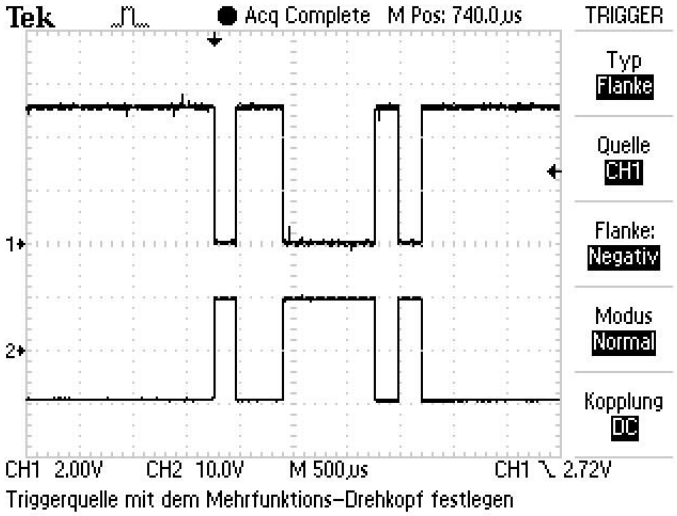
$$TH1 = 256 - \frac{8{,}867\ MHz}{12} * \frac{2^0}{4800 * 32} = 251{,}2 \rightarrow 251$$

Dieser Wert musste im Programm eingetragen werden.

4.3 Programm

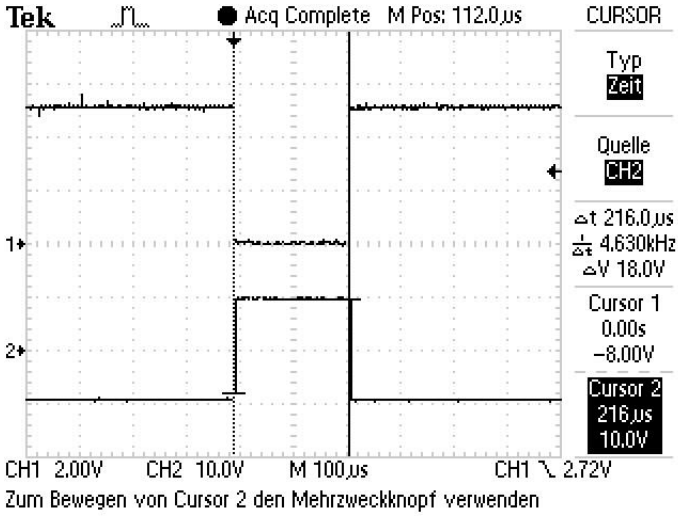
```
#include <at898252.h>
#include <stdio.h>

char Text[20];

void main (void)
{
      TMOD  = 0x20;               // Timer 1 im Mode 2
      TH1         = 251;          // Reload Value, 4800 Baud
      TL1         = 251;          // Timerinitialisierung
      SCON     = 0x50;            // UART Mode 1
      PCON     = PCON & 0x7F;     // SMOD-Bit = 0,
      TR1         = 1;            // Timer 1 starten
//    ES          = 1;            // Freigabe serieller Inetrrupt
//    EA          = 1;            // Freigabe Interrupt allgemein

      while (1)
      {
      TI  = 1;
      printf ("\nBitte einen Text eingeben.\n"); // \n...New Line
      TI  = 0;

      REN = 1;
      gets(Text,sizeof(Text));              // sizeof -> Stringlänge
      REN = 0;

      TI  = 1;
      printf("\nEingegeben wurde:%s\n",Text);         // %s...String
      TI  = 0;
      }
}
```

4.4 Messung des seriellen Signals



Bei dieser Messung ist am CH1 das serielle Signal vom Prozessor zu sehen, und am CH2 das Signal mit den an die RS232-Schnittstelle angepassten Pegel.
Der Prozessor liefert nur 0V und 5V, der MAX232 macht daraus +10V und -10V.

4.5 Messung der Bitdauer



Nun wurde in ein Bit hineingezoomt und die Bitdauer gemessen. Sie beträgt 216us.
Berechnet wurde ein Wert von 208,3us, was ziemlich gut mit der Messung übereinstimmt.

$$T_{Bit} = \frac{1}{Baudrate} = \frac{1}{4800} = 208{,}3us$$