Informatikwerkstatt, Foliensatz 3
PC-Kommunikation

G. Kemnitz

Institut für Informatik, TU Clausthal (IW-F3)

16. November 2015

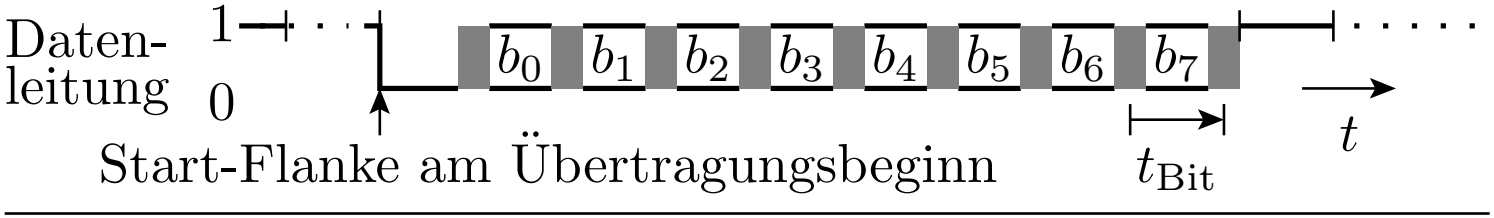
Inhalt Foliensatz 3

PC-Kommunikation

Protokoll

Kommunikationsprotokoll

Der Datenaustausch zwischen Rechnern erfolgt seriell[1] über USB, Ethernet, CAN-Bus, ... Unsere PC-Kommunikation nutzt USART2. Kommunikationsprotokoll[2] 8N1, 9600 Baud:



$b_i \in \{0,1\}$ Datenbits — Übertragungspause

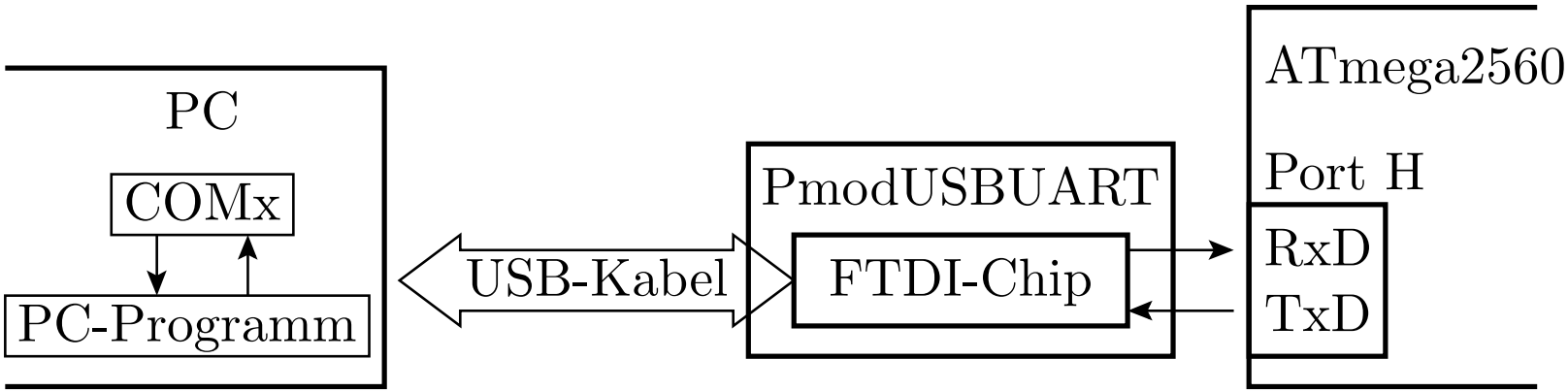
⊢⊣ Stoppbit (Wert 1) — Byteübertragung: $10 \cdot t_{\text{Bit}}$

- 8 Daten, kein Paritäts- und ein Stoppbit
- Bitzeit 1/9600s. (bis ca. 1000 Datenbytes pro s).

[1] Seriell: Hintereinander über eine, statt parallel über viele Leitungen.

[2] Kommunikationsprotokoll: Vereinbarung, nach der die Datenübertragung zwischen zwei oder mehr Teilnehmern erfolgt.

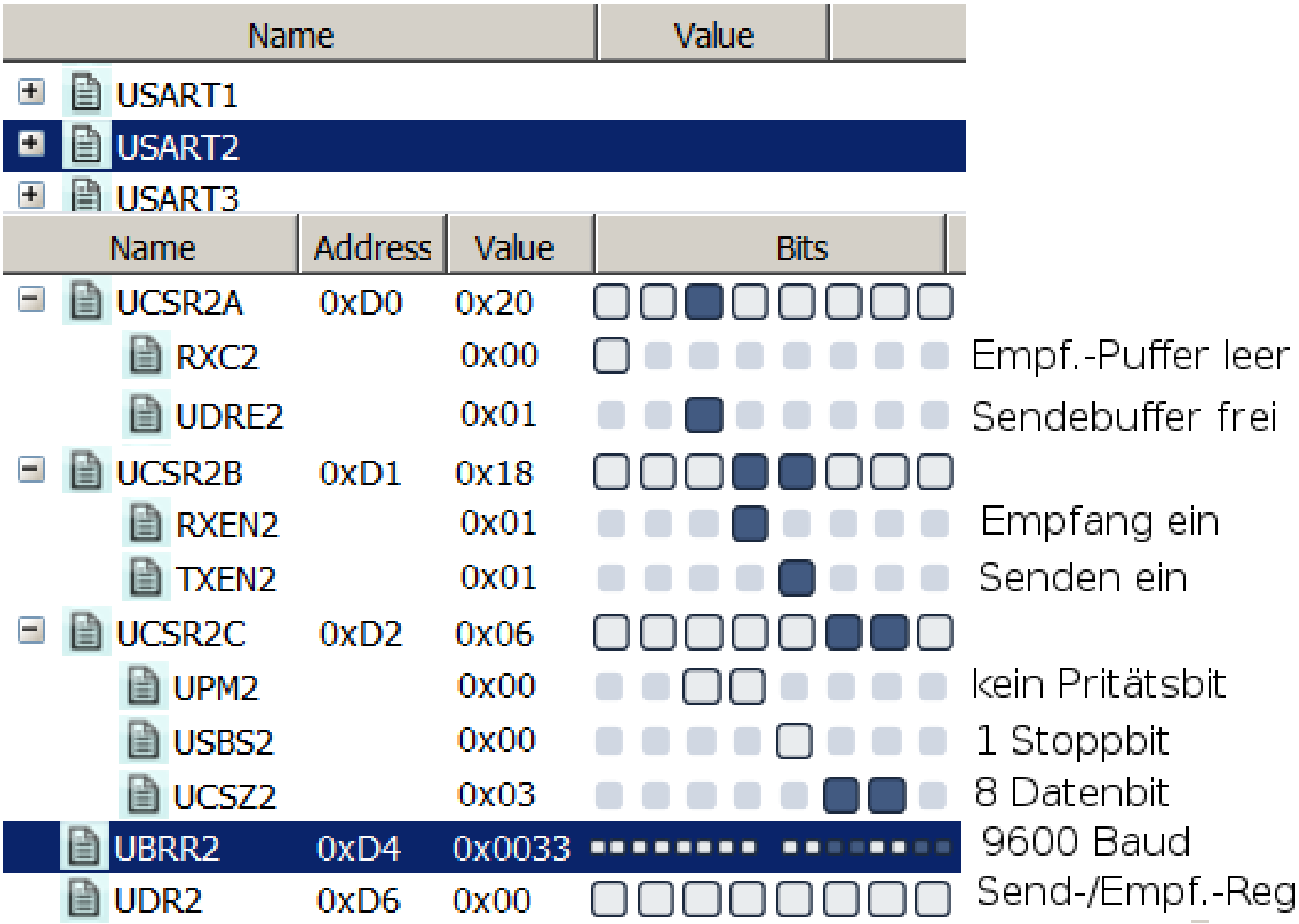
Kommunikationsfluss



- Der Mikrorechner kann zeitgleich einzelne Bytes zum FTDI-Chip senden und vom FTDI-Chip empfangen.
- Der FTDI-Chip tauscht über USB Daten mit dem PC aus.
- Auf dem PC präsentiert der Treiber für den FTDI-Chip die Datenverbindung zum Mikrorechner als COM-Port.
- Jeder einmal über USB verbundene FTDI-Chip bekommt einen eigenen COM-Port.
- Das PC-Programm wird entweder HTerm oder ein eigenes Python-Programm sein.

Byte-Empfang und Senden im Mikrorechner

```
int main()
  //Variablenvereinbarung und Initialisierung
  <USART2 Protokoll 8N1, 9600 Baud>
  <Sender und Empfänger ein>
  while (1){
    ...
    //Byteempfang
    <Warte, bis Byte da ist>
    <Lesen und Verarbeiten des Bytes>
    ...
    //Byte versenden
    <Warte, bis Sendepuffer frei>
    <Byte in Sendepuffer schreiben>
    ...
  }
}
```


Name
Value
USART1
USART2
USART3
Name
Address
Value
Bits
UCSR2A 0xD0 0x20
RXC2 0x00 Empf.-Puffer leer
UDRE2 0x01 Sendebuffer frei
UCSR2B 0xD1 0x18
RXEN2 0x01 Empfang ein
TXEN2 0x01 Senden ein
UCSR2C 0xD2 0x06
UPM2 0x00 kein Pritätsbit
USBS2 0x00 1 Stoppbit
UCSZ2 0x03 8 Datenbit
UBRR2 0xD4 0x0033 9600 Baud
UDR2 0xD6 0x00 Send-/Empf.-Reg

Echoprogramm

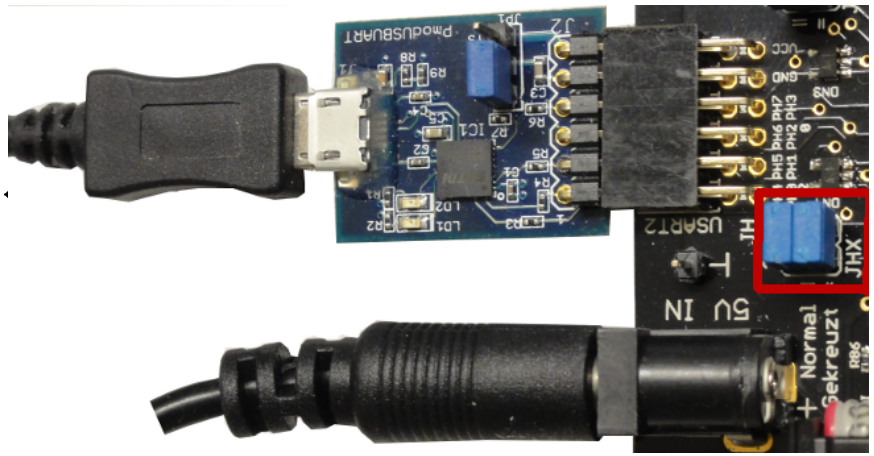
Das Echoprogramm für nachfolgende Experimente

```
#include <avr/io.h>
int main(void){                          //Projekt F3-1_echo
 UCSR2C=0b110;                           //Übertragungsformat 8N1
 UBRR2=51;                               //9600 Baud
 UCSR2B=(1<<RXEN2)|(1<<TXEN2);//Empf. + Sender ein
 DDRJ = 0xFF;                            //LEDs als Ausgänge
 uint8_t daten, Ct=0;                    //Zähler LED-Ausg.
 while(1){
  while (!(UCSR2A & (1<<RXC2)));//warte auf Byte
  daten = UDR2;                            //Byte übernehmen
  while (!(UCSR2A &(1<<UDRE2)));//warte Puffer frei
  UDR2 = daten;                            //Byte übergeben
  PORTJ=Ct;                                //Zählausgabe auf
  Ct++;                                    //LED erhöhen
 }
}
```

Test des Echo-Programms

Hardware vorbereiten:

- Spannung ausschalten.
- PModUSBUSART an JH oben anstecken.
- Jumper JHK »gekreuzt (=)«
- PModUSBUSART mit PC verbinden.
  Spannung einschalten.

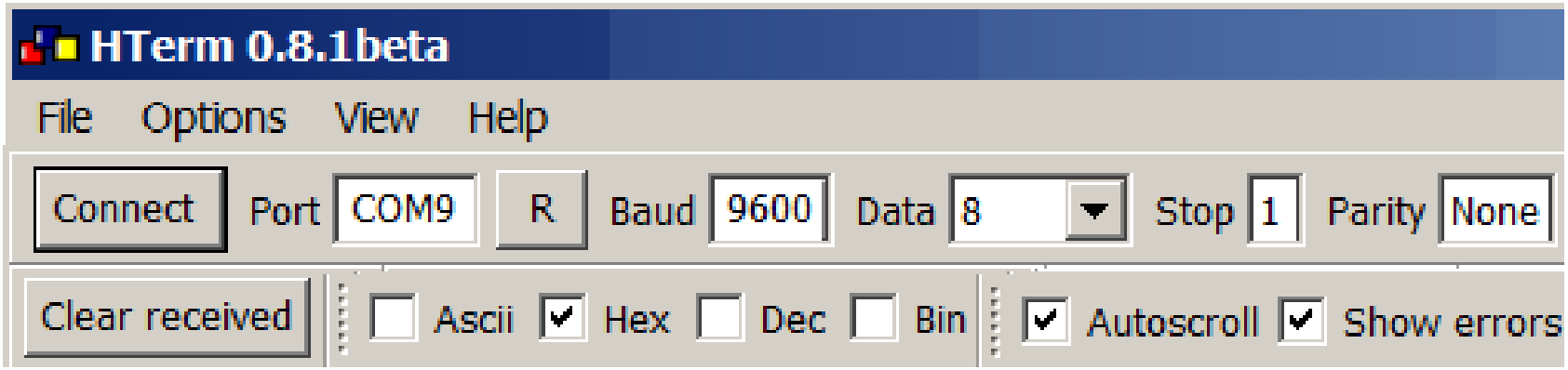


Software vorbereiten:

- Projekt Echo öffnen.
- »Dragon« und Compileroptimierung »-O0« auswählen.
- Übersetzen und im Debugger starten.

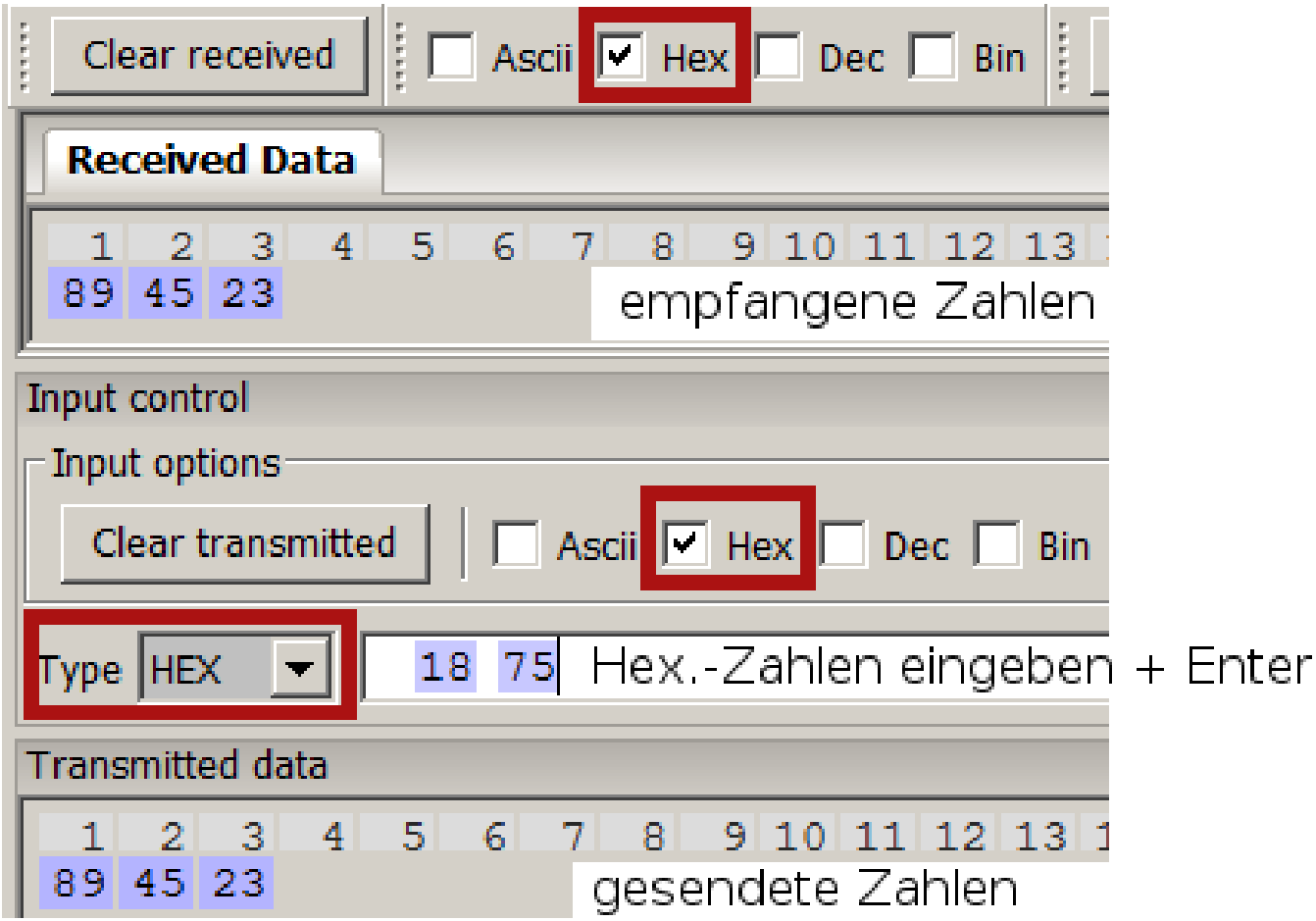
Verbindung mit HTerm herstellen

- Auf dem PC HTerm starten.
- COM-Port auswählen[3].
- 9600 Baud, 8 Daten, 1 Stopp- und kein Paritätsbit einstellen.
- Verbindung herstellen (Connect).



[3] Die COM-Schnittstelle, die nach Anstecken des USB-Kabels vom PmodUSBURT und »R« (Refresh Comport List) als neuer Port erscheint.

Für die Eingabe »HEX« auswählen. Für die Darstellung der Sende- und Empfangsdaten nur bei »Hex« $\surd$ setzen.



- Alle versendeten Zahlen werden zurückgesendet.

Textdarstellung

Zeichendarstellung im ASCII-Code

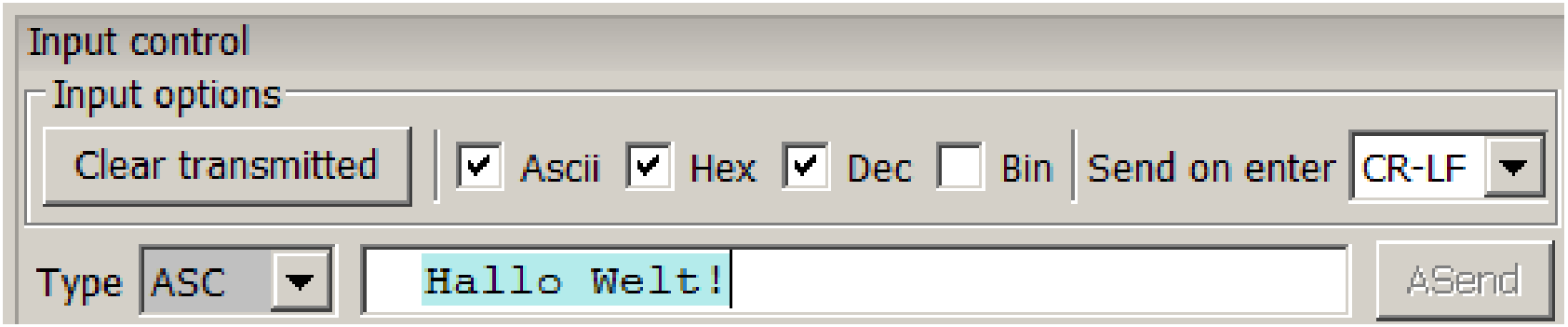
Buchstaben, Ziffern und andere Zeichen werden als Bytes und Texte als Felder von Bytes dargestellt. ASCII-Code:

hex	bin	dez	ASCII	hex	bin	dez	ASCII
0x0a	0b000 1010	10	LF	0x41	0b100 0001	65	A
0x0d	0b000 1101	13	CR	⋮	⋮	⋮	⋮
0x20	0b010 0000	32	␣	0x50	0b101 0000	80	P
0x21	0b010 0001	33	!	⋮	⋮	⋮	⋮
0x2E	0b010 1110	46	.	0x5A	0b101 1010	90	Z
0x30	0b011 0000	48	0	0x61	0b110 0001	97	a
0x31	0b011 0001	49	1	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	0x70	0b111 0000	112	p
0x39	0b011 1001	57	9	⋮	⋮	⋮	⋮
0x3F	0b011 1111	63	?	0x7A	0b111 1010	122	z

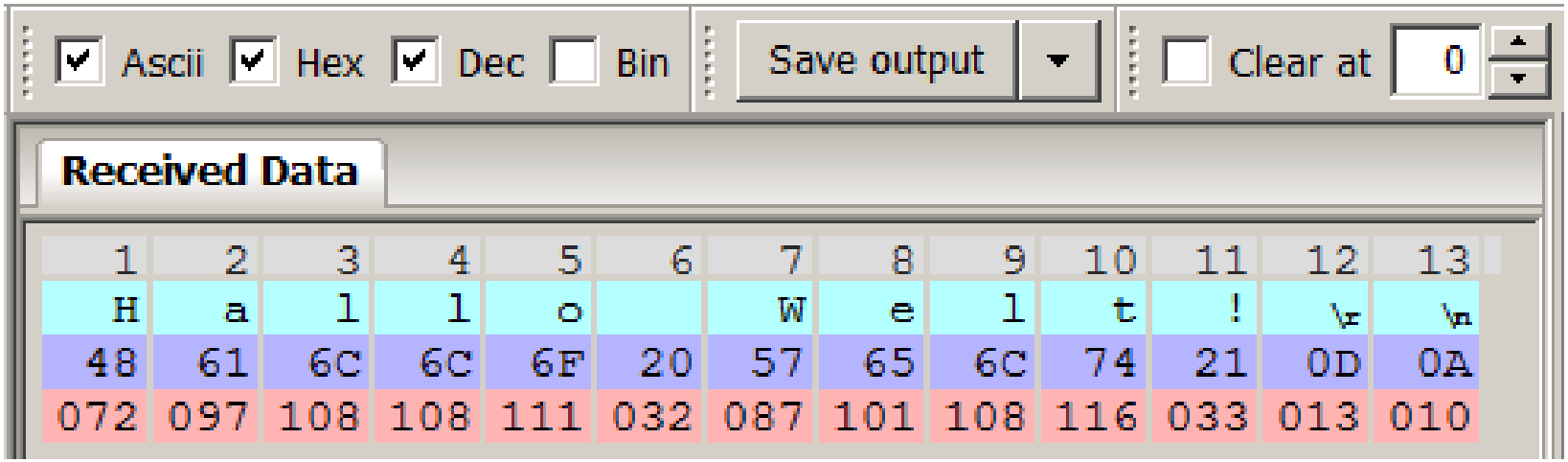
LF – Zeilenvorschub; CR – Wagenrücklauf; ␣ – Leerzeichen

Senden und Empfang von Texten

Das HTerm kann ASCII-Zeichenketten + CR+LF senden:



Empfangene Daten als Zeichen- und Zahlenfolge:



1	2	3	4	5	6	7	8	9	10	11	12	13
H	a	l	l	o		W	e	l	t	!	\r	\n
48	61	6C	6C	6F	20	57	65	6C	74	21	0D	0A
072	097	108	108	111	032	087	101	108	116	033	013	010

Kontrollieren Sie auch, dass sich bei jedem Senden der Led-Ausgabewert an LED1 bis LED8 um die Anzahl der gesendeten Zeichen erhöht.

Test mit Python

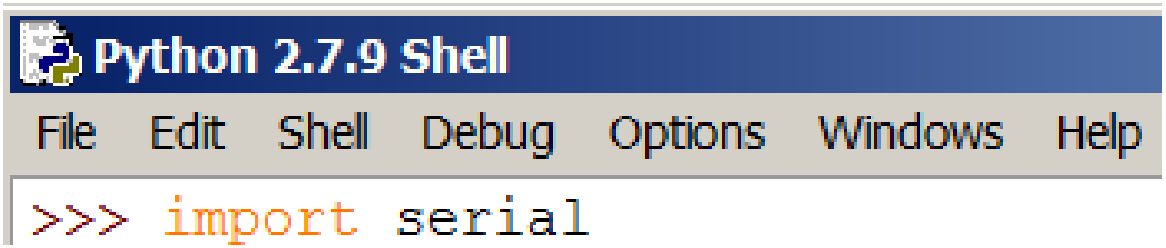
Python als Programmiersprache für Tests

Python: Interpretersprache, einfach zu erlernen, einfach zu programmieren, kurze Programme, gut zu debuggen, ...

- Verbindung mit HTerm schließen (Disconnect)
- Start der Programmierkonsole »Idle«

  Start > All Programs > Python(x,y) > IDLE

- Import des Moduls für die serielle Kommunikation:



- Kommunikationsverbindung öffnen:[4]

```
ser = serial.Serial("COM9")
```

[4] Dieselbe COM wie im HTerm benutzen. 8N1, 9600 Baud ist der Standardwert und muss deshalb nicht eingestellt werden.

- Zeichenfolge an Variable zuweisen:

```
send = "Hallo Welt!"
```

- Anschauen von Typ, Wert und Länge der Zeichenkette:

`>>> type(send)` `<type 'str'>`	Funktion zur Bestimmung des Datentyps. Der Datentyp ist "Zeichenkette (string)".
`>>> send` `'Hallo Welt!'`	Bei Eingabe des Variablennamens wird der Wert angezeigt.
`>>> len(send)` `11`	Die Funktion "len()" liefert die Anzahl der Elemente der Zeichenkette.

- Zeichenkette senden:

```
ser.write(send)
```

- Auf 11 Zeichen warten und diese lesen:

```
y = ser.read(11)
```

- Anschauen von Typ, Wert und Länge der empfangenen wie bei den gesendeten Daten.

Zusammenfassen zu einem Programm (scom.py)

- Programmdatei in der »Idle« öffnen:

  ```
  File > Open > H:~\Informatikwerkstatt\Pythom\scom.py
  ```

  Das Programm:

  ```
  import serial
  ser = serial.Serial("COM9")
  send = "Hallo Welt!"
  ser.write(send)
  y = ser.read(len(send))
  print "Empfangene Daten:", y
  ser.close()
  ```

- »print« dient der Ausgabe
- In Python müssen alle Zeilen eines Blocks gleiche Einrücktiefe haben.
- Programmstart mit Ausgabe auf der »Idle«:

  ```
  Run > Run Module (F5)
  ```

Start auf der Konsole

- Konsole öffnen (Eingabe von »cmd + Enter« im Suchfeld über »Start«):

- Wechsel in das Programmverzeichnis. Eingabe Programmnamen + Enter:

```
C:\Windows\system32\cmd.exe
C:\>H:
H:\>cd Informatikwerkstatt\Python
H:\Informatikwerkstatt\Python>scom.py
Empfangene Daten: Hallo Welt!
```

- Die Programmausgabe »Empfangene Daten: Hallo Welt!« erfolgt auf der Konsole.

Messung der Übertragungsdauer

```
#Programmname: Python/scom_t.py
import serial
# Funktion "clock() aus Modul "time" importieren
from time import clock
ser = serial.Serial("COM9")
send = "Hallo Welt!"
ts = clock();      #Startzeit in Sekunden nach ...
ser.write(send)
y = ser.read(len(send))
dt = clock()-ts    #Zeitdifferenz zur Startzeit
print "Empfangene Daten:", y, "dt=", dt*1000, "ms"
ser.close()
```

Programmstart und Ausgabe auf der Konsole:

```
H:\Informatikwerkstatt\Progr_IW\Python>scom_t.py
Empfangene Daten: Hallo Welt! dt= 13.6730654185 ms
```

Paketgröße und Übertragungszeit (scom_txy.py)

```
import serial
from time import clock
ser = serial.Serial("COM9", timeout=1)
send = "Das ist ein sehr langer String!"
l = 1 dt_list=[];     #leere Liste fuer dt-Werte
while l<=len(send):   #Wiederhole bis Gesamtlaenge
  ts = clock();       #neuer Block => einruecken
  ser.write(send[:l])#Sende die ersten l Zeichen
  y = ser.read(l)     #Warte auf l empfangene Zeichen
  dt = clock()-ts     #Zeitdifferenz zur Startzeit
  dt_list.append(dt) #Differenzzeiten an Liste haengen
  print "Empfangene Daten:", y, "dt=", dt*1000, "ms"
  l = l+1;            #Ende des Schleifenkoerpers
ser.close()           #danach Einruecktiefe ruecksetzen
```

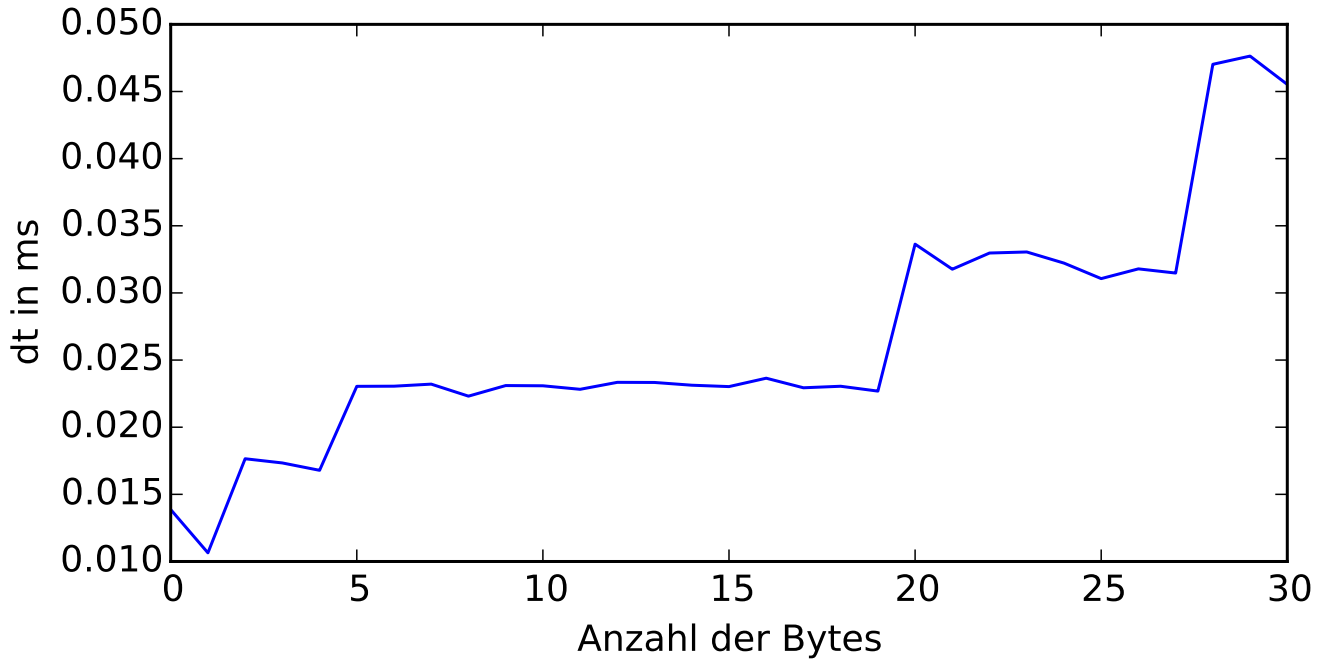
Das Programm bestimmt für Bytefolgen der Länge 1 bis 30 die Übertragungsdauer. Ausprobieren!

Programmstart und Ausgabe auf der Konsole

```
H:\Informatikwerkstatt\Progr_IW\Python>scom_txy.py
Empfangene Daten: D dt= 13.2549278489 ms
Empfangene Daten: Da dt= 14.921187855 ms
Empfangene Daten: Das dt= 14.4043923316 ms
Empfangene Daten: Das  dt= 14.4265738178 ms
Empfangene Daten: Das i dt= 14.3878389836 ms
Empfangene Daten: Das is dt= 14.2295889771 ms
Empfangene Daten: Das ist dt= 14.368306033 ms
Empfangene Daten: Das ist  dt= 14.3656574973 ms
Empfangene Daten: Das ist e dt= 14.2249540396 ms
Empfangene Daten: Das ist ei dt= 14.1339106258 ms
Empfangene Daten: Das ist ein dt= 14.1051078004 ms
Empfangene Daten: Das ist ein  dt= 14.4951046784 ms
Empfangene Daten: Das ist ein s dt= 30.4876252137 ms
Empfangene Daten: Das ist ein se dt= 29.8907114861 ms
Empfangene Daten: Das ist ein seh dt= 30.3879740589 ms
Empfangene Daten: Das ist ein sehr dt= 30.0079091897 ms
Empfangene Daten: Das ist ein sehr  dt= 30.3273888054 ms
Empfangene Daten: Das ist ein sehr l dt= 30.1208030228 ms
Empfangene Daten: Das ist ein sehr la dt= 30.1522543839 ms
Empfangene Daten: Das ist ein sehr lan dt= 30.1446398438 ms
Empfangene Daten: Das ist ein sehr lang dt= 30.2389939272 ms
Empfangene Daten: Das ist ein sehr lange dt= 30.3604955013 ms
```

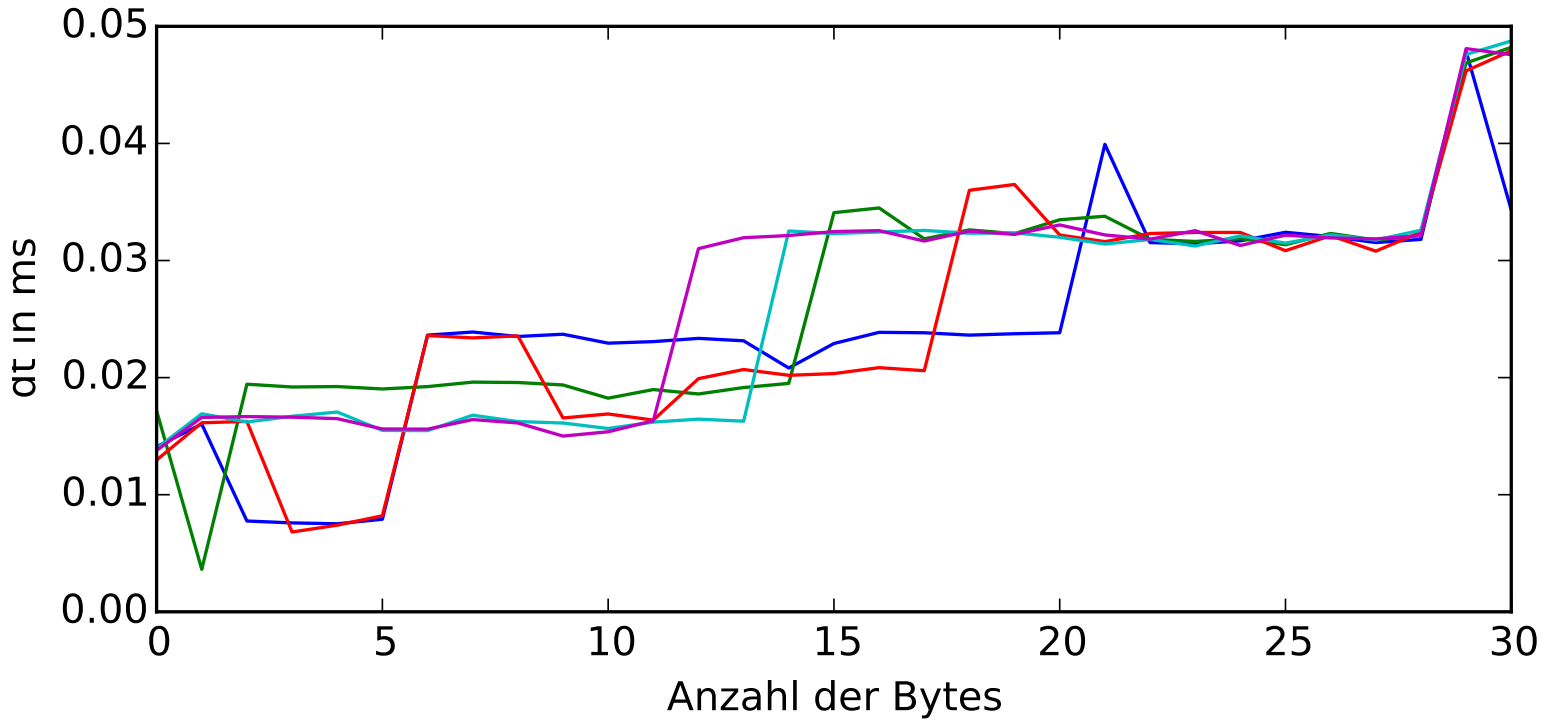
Graphische Darstellung der Liste der Zeitdifferenzen:

```
# Import der Klasse "pyplot"
import matplotlib.pyplot as plt
plt.plot(range(len(dt_list)), dt_list)
plt.xlabel("Anzahl der Bytes")  #plot erzeugen
plt.ylabel("dt in ms")          #Achsen beschriften
plt.show()                      #anzeigen
```



Weder deterministisch noch linear. Grund Tunnelung durch USB.

Das Programm »scom_txy5.py« führt die Messungen zusätzlich 5x in einer Schleife aus. Beispielergebnis:



Bei 9600 Bitzeiten pro s, 8 Datenbits + 1 Startbit dauert eine Byteübertragung ca. 1 ms. Für 21 bis 28 Byte große Pakete werden etwa 30 ms benötigt, d.h. fast max. Durchsatz.

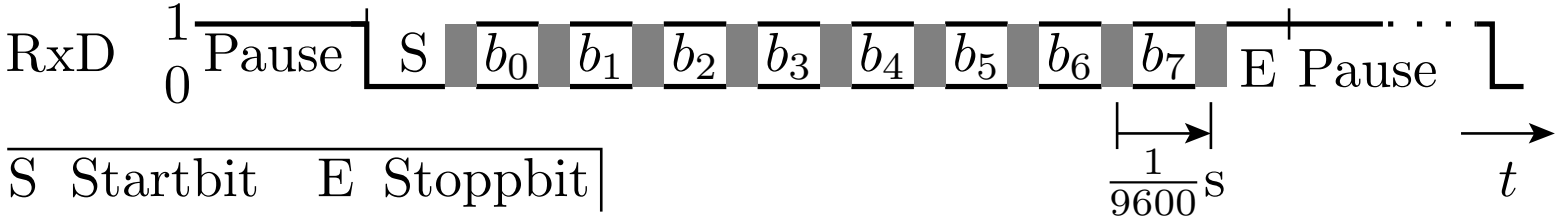
Test mit Logikanalysator

Test mit Logikanalysator

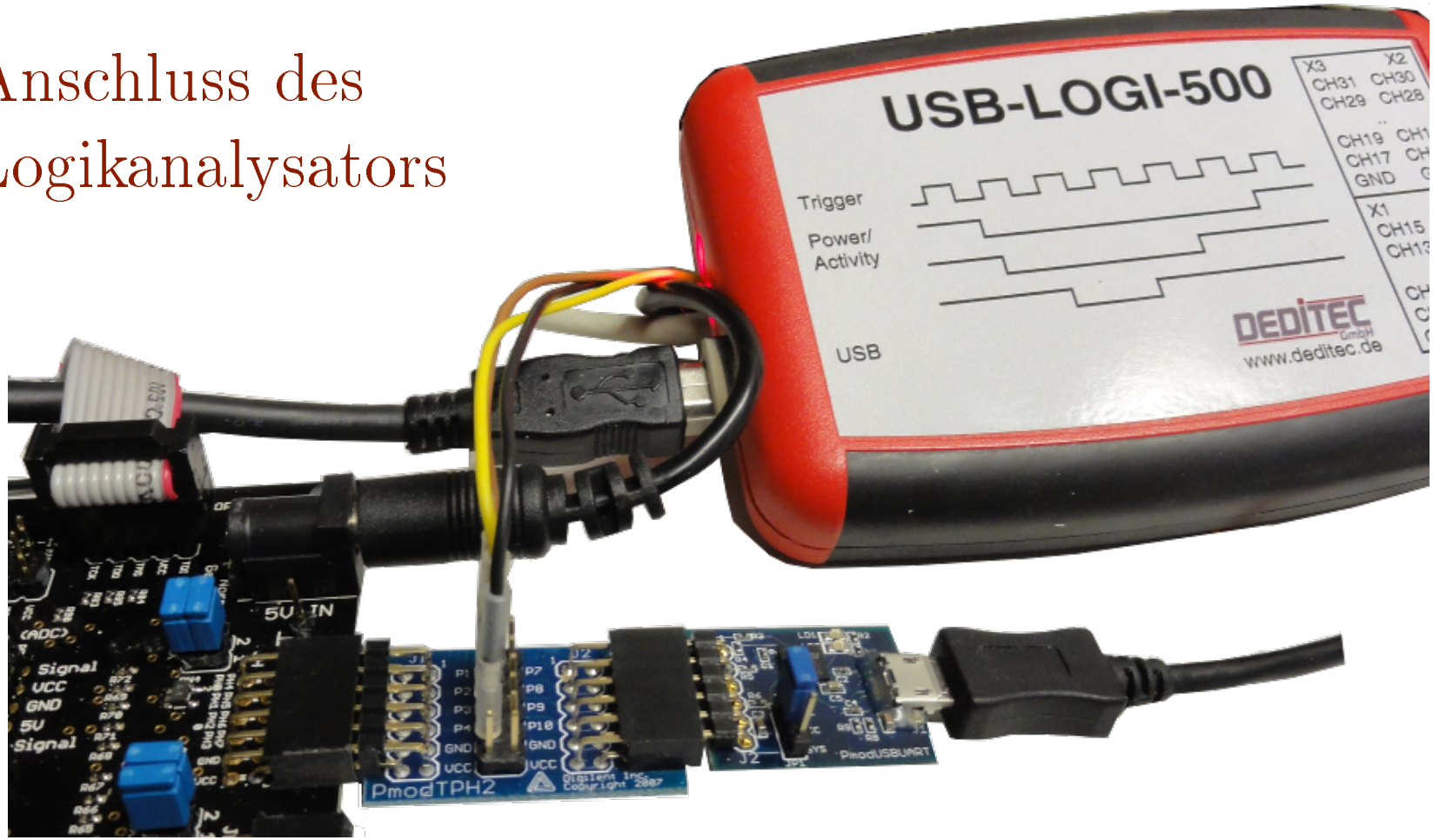
Ein Logikanalysator zeichnet Logikwerte mit einem vorgegebenen Aufzeichnungstakt auf.
Die Auswahl des Aufzeichnungsfensters erfolgt über

- Trigger: logische Bedingung bezüglich der Signalwerte.
- Pre-Trigger: Anteil der vor der Triggerbedingung aufzuzeichnenden Werte

Für die Experimente dient ein USB-Logi. Trigger sei die erste fallende Flanke auf der Empfangsleitung »RxD«, Pre-Trigger der Standdardwert »1/8 des Aufzeichnungsspeichers«.
Aufzeichnungstakt 50 kHz (5 Werte pro Bitzeit).



Anschluss des Logikanalysators



PModUSBUSART über Zwischenadapter PmodTPH2 anstecken.

- GND (schwarz) an Masse (Adapter Gnd)
- ch0 (gelb) an RxD (Adapter P3) anstecken
- ch1 (braun) an TxD (Adapter P2) anstecken.

Konfiguration des USB-Logi

Der USB-Logi wird bei uns mit einem xml-File, dass auf der Kommandozeile übergeben wird, konfiguriert (siehe Doku auf der Web-Seite). Für das Beispiel:

- xml-File »echotest.xml« zur Steuerung der Aufzeichnung:

```
<la>
  #Aufzeichnung von 50.000 Werte pro Sekunde
  <samplerate>50000</samplerate>
  <signals>
    <signal name="RxD"> <ch>0</ch> </signal> #RxD => CH0
    <signal name="TxD"> <ch>1</ch> </signal> #TxD => CH1
  </signals>
  #Aufzeichnungsbeginn 1/8 der Gesamtaufzeichnungsdauer
  #vor der fallenden Flanke von RxD
  <trigger when="A">
    <A> <ch when="falling_edge">0</ch> </A>
  </trigger> </la>
```

Messung durchführen

- Weitere Windows-Konsole (cmd.exe) starten.
- Zum Verzeichnis »H:Informatikwerkstatt\USBLOGI« wechseln und USB-Logi starten mit dem Kommando:

```
usb-logi echotest.xml
```

```
H:\Informatikwerkstatt\USBLOGI>usb-logi echotest.xml
+============================================================+
! USB-LOGI-500 Logik Analysator                              !
+============================================================+
        xml-Datei prüfen, Verbindung herstellen, ...
Fülle Pre-Trigger...

                                                          100 %

Warte auf Trigger...
```

- Sobald der USB-Logi auf sein Triggerereignis wartet, mit der ersten Windows-Konsole »scom_txy« starten.

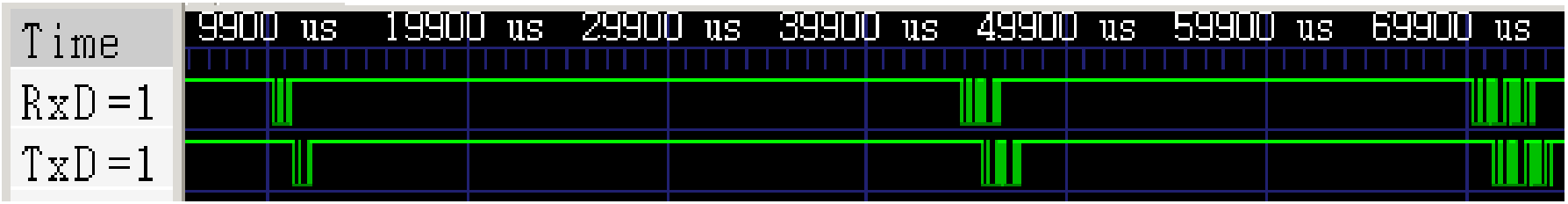
```
H:\Informatikwerkstatt\Progr_IW\Python>scom_txy.py
Empfangene Daten: D dt= 13.2549278489 ms
Empfangene Daten: Da dt= 14.921187855 ms
Empfangene Daten: Das dt= 14.4043923316 ms
Empfangene Daten: Das  dt= 14.4265738178 ms
                  . . .
```

- Bei der ersten fallenden Flanke beginnt die Aufzeichnung.

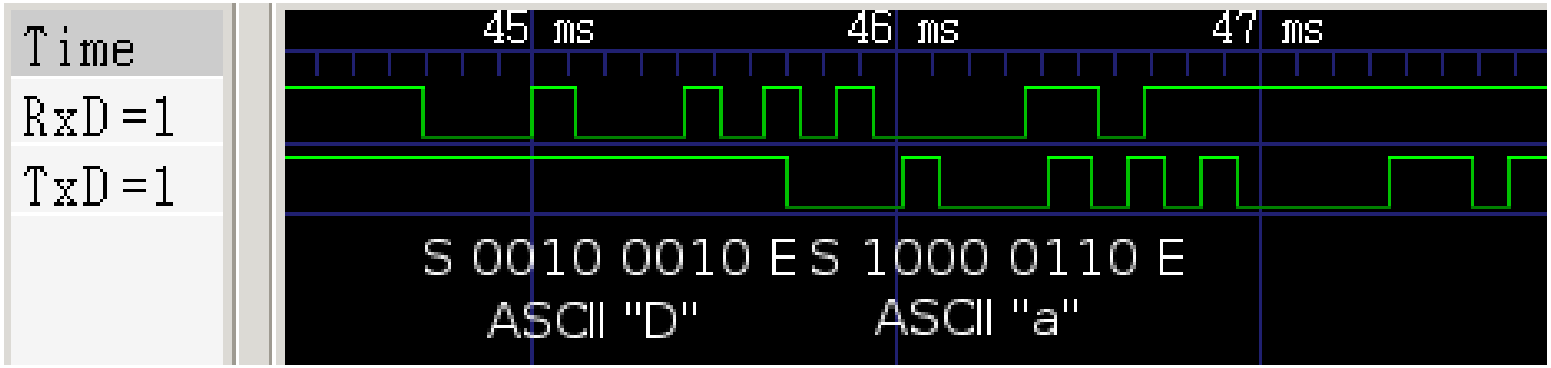
```
Warte auf Trigger...
Aufzeichnung läuft...
                                                  100 %
+-----------------------------------------------------+
| (4) Erzeuge Waveform Dateien...                     |
+-----------------------------------------------------+
LXT Datei ./echotest.lxt erzeugt.
Bestehende SAV Datei ./echotest.sav beibehalten.
+-----------------------------------------------------+
| (5) Starte GTKWave...                               |
+-----------------------------------------------------+
```

- Entspechend Einstellung werden 1/8 der Werte vor. und 7/8 der Werte nach dem Trigger-Ereignis aufgezeichnet.

Im sich öffnenden GTKWave-Fenster mit Zoom und Scroll-Leisten Darstellung anpassen.



1-Byte-Paket nach ca. 10 ms, 2 Byte-Paket nach ca. 50 ms und 3-Byte-Paket nach ca. 75 ms. Vergrößertes 2-Byte-Packet:



Übertragene Zeichenfolge »Da«. Mikrorechner beginnt mit Rückübertragung nach halber Stoppbitzeit.

Modularisierung

Modularisierung des Echoprogramms

```
#include <avr/io.h>
int main(void){
  //Funktion void com_pc_init()
  UCSR2C=0b110;                    //Uebertrag.-Format 8N1
  UBRR2=51;                        //9600 Baud
  UCSR2B=(1<<RXEN2)|(1<<TXEN2);//Empfaenger und Sender
  //-------------------------------------------------
  while(1){
    //Funktion uint8_t getByte()
    while (!(UCSR2A & (1<<RXC2)));//warte auf ein Byte
    daten = UDR2;                  //Byte uebernehmen
    //-------------------------------------------------
    //Funktion void sendByte(uint8_t dat)
    while (!(UCSR2A &(1<<UDRE2)));//warte Puffer frei
    UDR2 = daten;                  //Byte uebergeben
    //-------------------------------------------------
  }
}
```

Funktionssammlung »F3-2_com_pc.c«

```
#include <avr/io.h>
// Initialisierung von USART2 (Protokoll 8N1, 9600 Baud)
void com_pc_init(){
  UCSR2C=0b110;                    //Uebertragungsformat 8N1
  UBRR2=51;                        //9600 Baud
  UCSR2B=(1<<RXEN2)|(1<<TXEN2); //Empf. und Sender ein
}

//ein Byte empfangen
uint8_t getByte(){
  while (!(UCSR2A & (1<<RXC2)));//warte auf ein Byte
  return UDR2;                     //Byte zurueckgeben
}

//ein Byte versenden
void sendByte(uint8_t dat){
  while (!(UCSR2A & (1<<UDRE2)));//warte Puffer frei
  UDR2 = dat;                       //Byte uebernehmen
}
```

Header mit den Funktionsdefinitionen

```
#ifndef COM_PC_H_
#define COM_PC_H_
 void com_pc_init();          //USART2 initialisieren
 uint8_t getByte();           //Byte empfangen
 void sendByte(uint8_t dat); //Byte versenden
#endif /* COM_PC_H_ */
```

Die Rahmenstruktur

```
#ifndef COM_PC_H_
#define COM_PC_H_
  ...
#endif
```

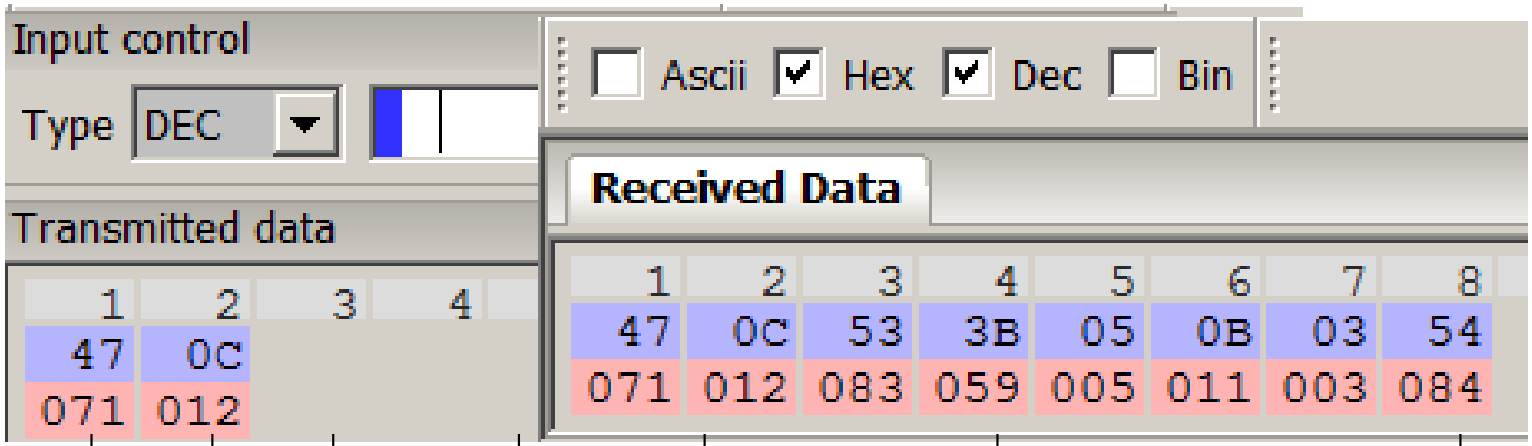
verhindert Mehrfachdefinitionen bei mehrfachem Einfügen des Headers. Wird als Template vorgegeben, ist vernünftig, aber nicht zwingend.

Testprogramm für arithmetische Operationen

```
#include <avr/io.h>          //test_com_pc.c
#include "com_pc.h"
uint8_t a, b, s, d, q, r;
uint16_t p; int main(){
  com_pc_init();
  while (1){
    a = getByte();  b = getByte();
    s = a + b;     //Summe
    d = a - b;     //Differenz
    p = a * b;     //Produkt
    q = a/b;       //ganzzahliger Quotient
    r = a%b;       //Divisionsrest
    sendByte(a);     sendByte(b);
    sendByte(s);     sendByte(d);
    sendByte(q);     sendByte(r);
    sendByte(p>>8); sendByte(p&0xFF);
  }
}
```

Test mit dem HTerm

- Projekt »F3-2_scom« öffnen.
- »Dragon« und Compiler-Optimierung -O0 auswählen.
- Übersetzen. Debugger starten. Programmstart.
- HTerm öffnen. 9600 Baud, 8 Datenbit, 1 Stoppbit.
- COM-Port des angesteckten »PmodUSBUART«. »Connect«.
- 2 Byte senden und 8 Bytes empfangen.



a	b	$a+b$	$a-b$	a/b	$a \cdot b$
71	12	83	59	5 Rest 11	$3 \cdot 2^8 + 84 = 852$

Python-Programm für den Test vom PC aus

```
import serial                    #Programm: test_com_pc.py
ser = serial.Serial("COM9")#COM anpassen!
#Testbeispiele
i_tuple = ((27,87),(220,20),(110,4), (218, 219))
for inp in i_tuple:              #fuer alle Testbeispiele
  ser.write(chr(inp[0])+chr(inp[1]))
  x = ser.read(8)
  a = ord(x[0]); b = ord(x[1]); s = ord(x[2])
  d = ord(x[3]); q = ord(x[4]); r = ord(x[5])
  p = ord(x[6]) * 256 + ord(x[7])
  print 'a=%3i'%a, 'b=%3i'%b, 'Summe:%i'%s, s==a+b
  print ' '*11, 'Differenz:%i'%d, d==a-b
  print ' '*11, 'Quotion:%i'%q, 'Rest:%i'%r, a==b*q+r
  print ' '*11, 'Produkt:%i'%p, p==a*b
ser.close()
```

chr() – Umwandlung Zahl => Zeichen; ord() – Umwandlung Zeichen => Zahl; ..==.. – Vergleichsergebnisse (False oder True)

Testdurchführung

- HTerm »Disconnect«.
- Auf dem Mikrorechner muss Programm »test_scom« laufen.
- Windows-Konsole (cmd.exe) starten. Im Verzeichnis H:\Informatikwerkstatt\Python das Programm test_com_pc.py starten. Programmausgabe kontrollieren::

```
H:\Informatikwerkstatt\Progr_IW\Python>test_com_pc.py
a= 27 b= 87 Summe: 114 True
            Differenz: 196 False
            Quotion: 0 Rest: 27 True
            Produkt: 2349 True
a=220 b= 20 Summe: 240 True
            Differenz: 200 True
            Quotion: 11 Rest: 0 True
            Produkt: 4400 True
a=110 b=  4 Summe: 114 True
            Differenz: 106 True
            Quotion: 27 Rest: 2 True
            Produkt: 440 True
a=218 b=219 Summe: 181 False
            Differenz: 255 False
            Quotion: 0 Rest: 218 True
            Produkt: 47742 True
```

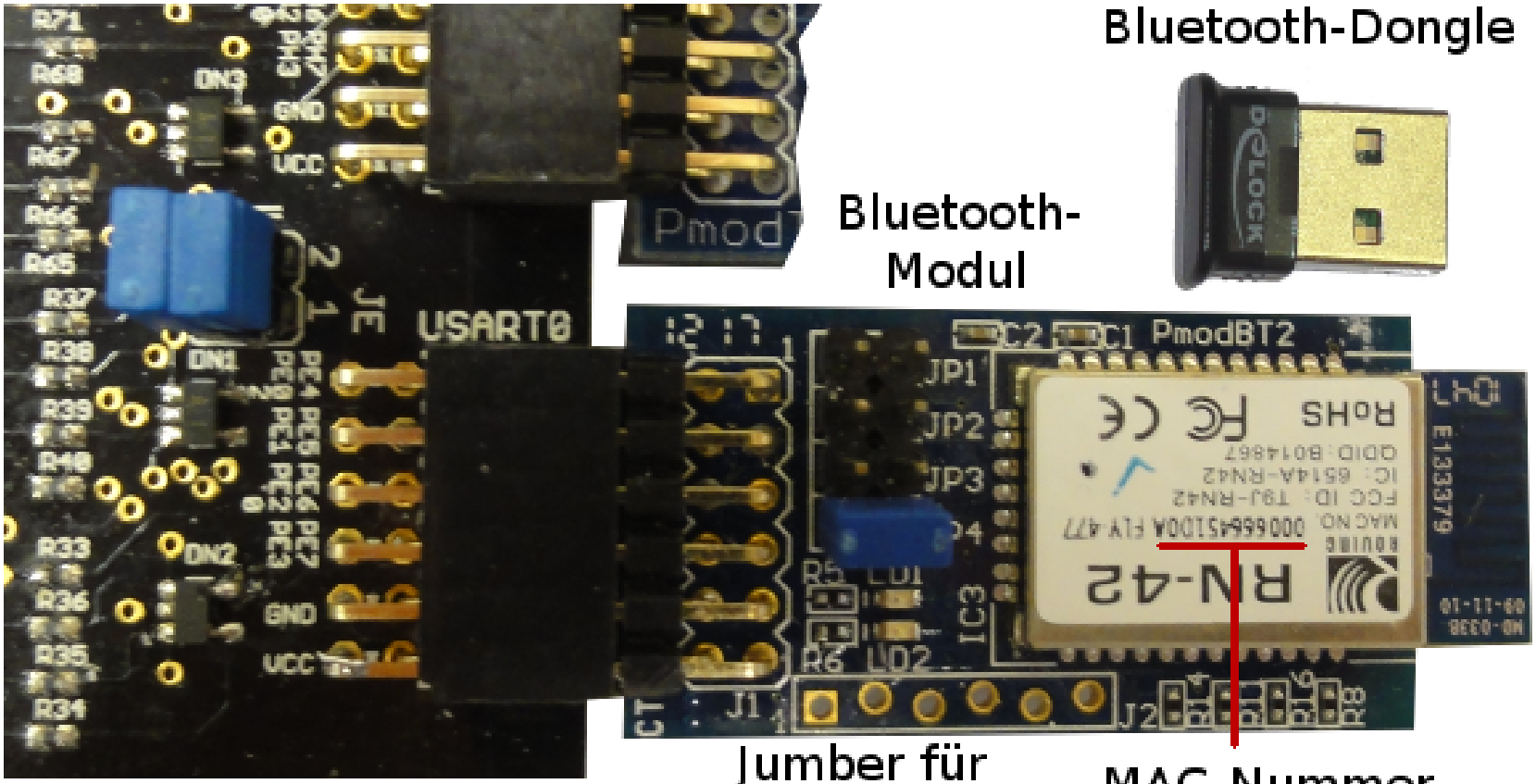
Bluetooth

Bluetooth

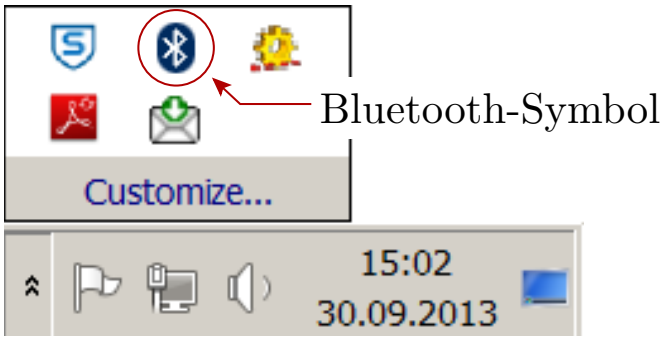
Bei Bluetooth wird die serielle Übertragung anstatt über eine USB- über eine Funkverbindung getunnelt.

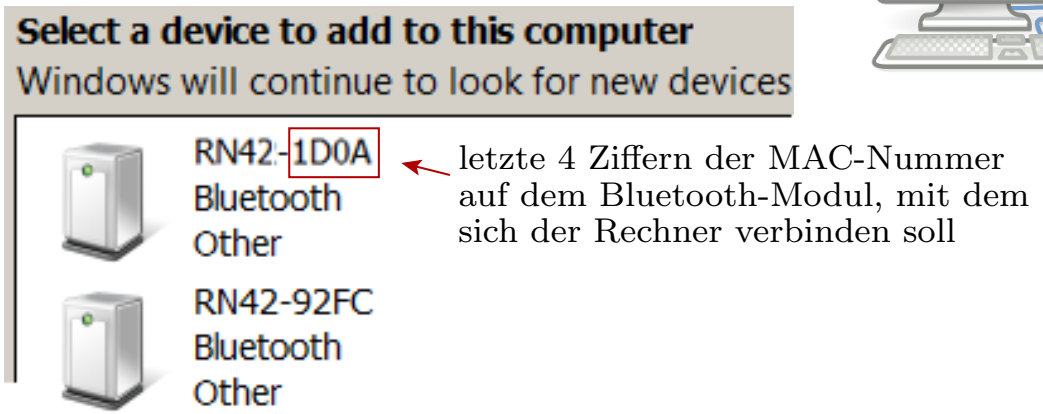
Bluetooth-Modul anschließen

- PmodBT2 an JE (USART0) stecken und Jumper JEX »gekreuzt (=)«.
- Bluetooth-Dongle in den PC stecken.



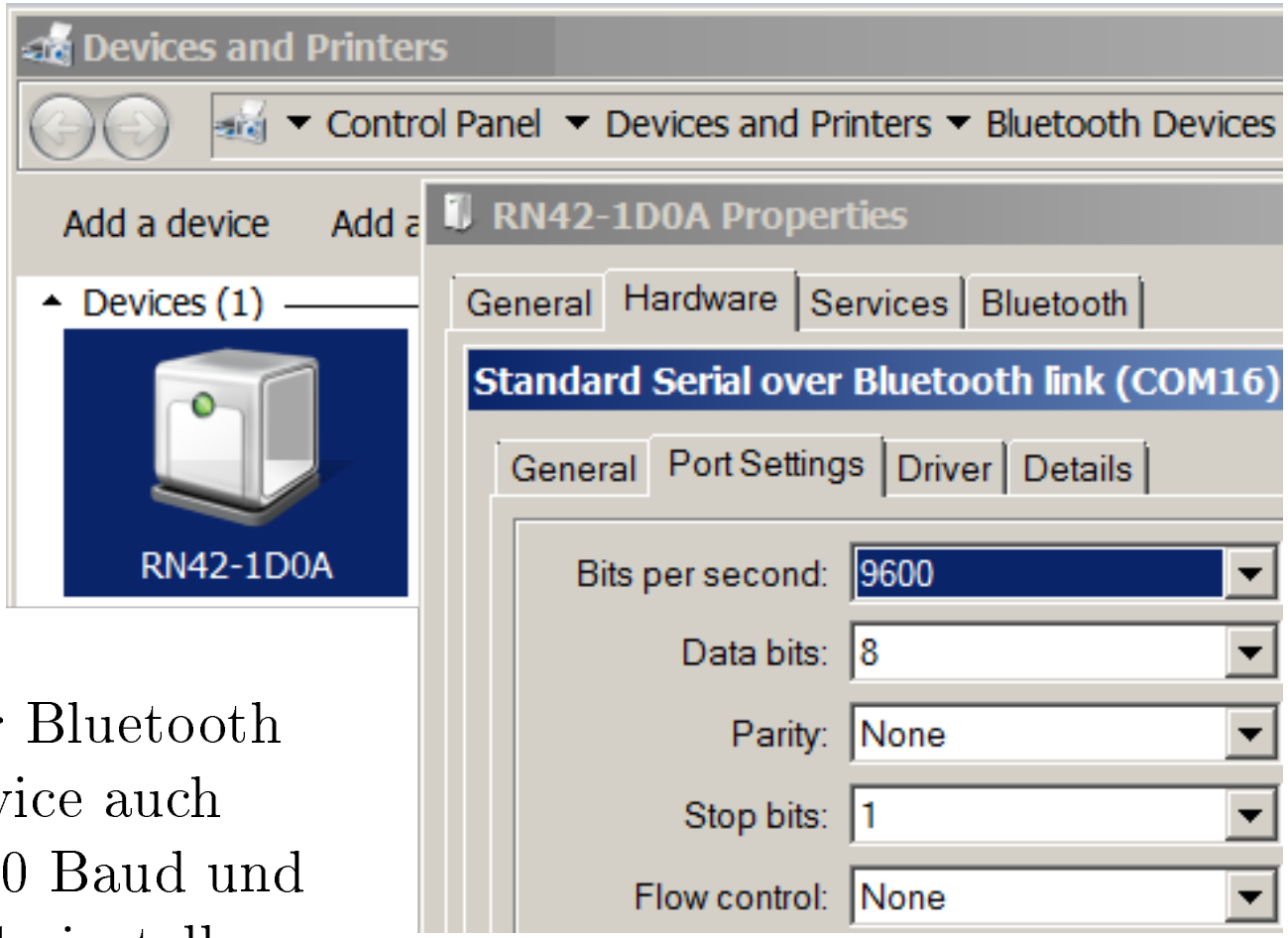
Bluetooth-Verbindung auf PC einrichten





- Unter Windows Doppelklick auf Bluetooth-Symbol
- Show Bluetooth Devices (Bei ersten mal »Add a Device«).
- Device mit der MAC-Nummer auf dem PMOD auswählen.
- Für »Enter the Device Pairing Code« Eingabe »1234«.[5]
- recher Mouse-Click > Properties > Hardware; hinter dem Namen COM-Port ablesen.
- HTerm: abgelesener COM-Port, 9600 Baud, 8N1, »connect«.

[5] Fehler, wenn zwischen Eingabeaufforderung und Eingabe zu lange gewartet wird.



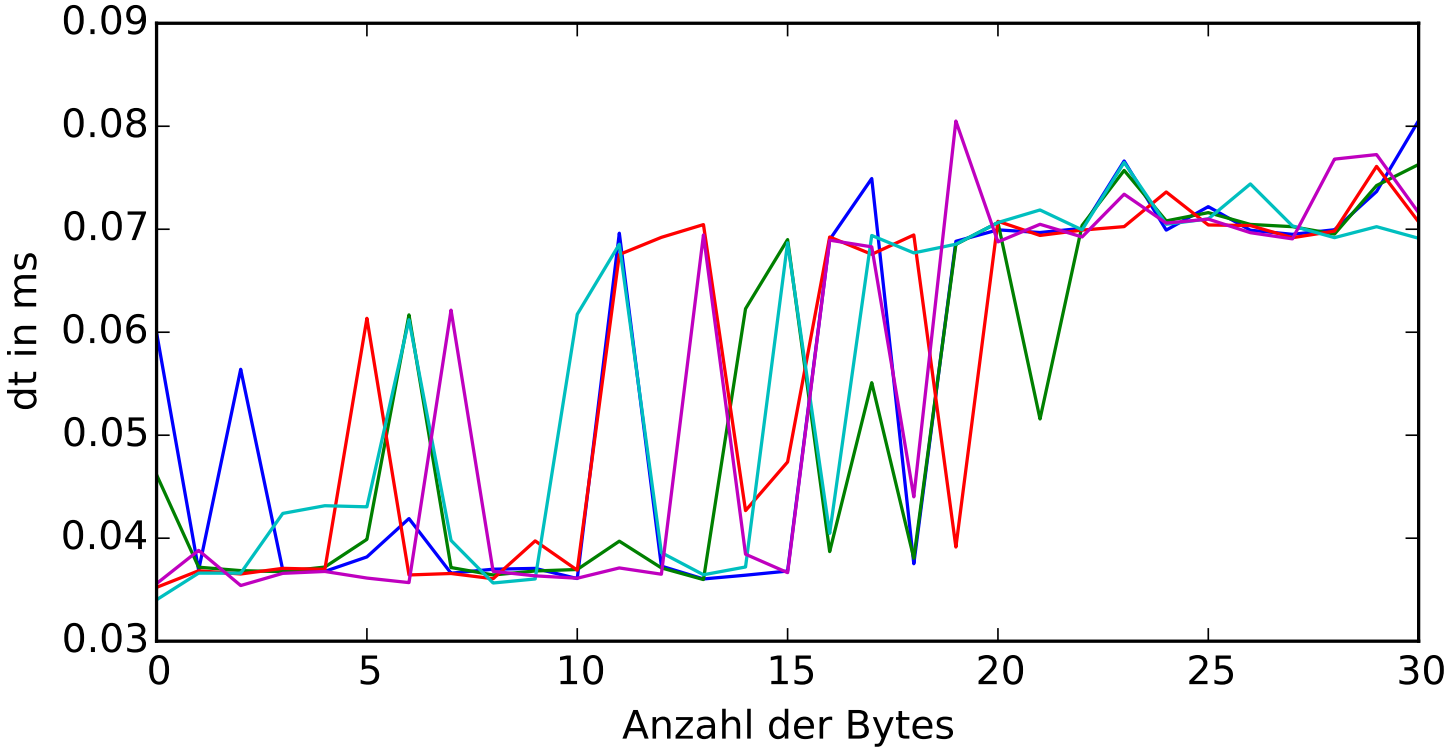
- Für Bluetooth Device auch 9600 Baud und 8N1 einstellen.
- In ATMEL-Studio Programm »echo_bt« starten.
- Mit HTerm Zeichen senden. Empfang + LED-Ausgabe der Zeichenanzahl kontrollieren.

Untersuchung der Übertragungsdauer

- Programm »echo_bt.c« weiter laufen lassen.
- Programm »Python\scom_txy5.py« im Editor öffnen. COM-Port durch den für die Bluetooth-Verbindung ersetzen.
- HTerm »Disconnect«.
- Windows-Konsole (cmd) starten. In das Verzeichnis ...\Python wechseln. Programm »scom_txy5.py« starten:

```
H:\Informatikwerkstatt\Python>scom_txy5.py
Empfangene Daten: D dt= 62.1397588383 ms
Empfangene Daten: Da dt= 60.5297850124 ms
Empfangene Daten: Das dt= 36.0593746958 ms
                  ...
```

- Graphische Darstellung der Zeitmesswerte nächste Folie.



- Die Übertragungsdauer hat einen zufälligen Wert.
- Für 21 bis 28 Byte große Pakete etwa doppelte Dauer (≈70 ms) im Vergleich zur Tunnelung durch USB.
- Wie ändert sich die Übertragungsdauer, wenn zeitgleich über mehrere Bluetooth-Verbindungen Daten übertragen werden?

Aufgaben

Aufgabe 3.1:

Durch welche Zahlenfolge wird der nachfolgende Text dargestellt:

`»Informatikwerkstatt, Uebung3«`

- Lösen Sie die Aufgabe mit der Tabelle auf Folie 15.
- Kontrollieren Sie das Ergebnis, in dem Sie die Zeichenkette mit dem HTerm versenden und zusätzlich als ASCII-Folge anzeigen lassen.
- Kontrollieren Sie das Ergebnis mit folgendem Programm:

```
#include <avr/io.h>
#include "com_pc.h"
uint8_t text[] = "Informatikwerkstatt, Uebung3";
int main(){
  com_pc_init();
  uint8_t i;
  for (i=0;i<28; i++) sendByte(text[i]);
}
```

Aufgabe 3.2:

Wenn C wie in der nachfolgenden Programmzeile

```
uint8_t text[] = "Informatikwerkstatt, Uebung3";
```

eine Zeichenkette initialisiert, hängt es ein Byte mit dem Wert null an.

- Kontrollieren Sie das mit dem Debugger.
- Schreiben Sie das Programm so um, dass es nicht genau 28 Zeichen ausgibt, sondern alle Zeichen bis zum Zeichenwert null.

  Hinweis: Man nutzt hierfür eine Schleife, »wiederhole, bis ein Zeiger »ptr« auf den Wert null zeigt:

  ```
  while (*ptr/=0){<Anweisungsfolge}
  ```

Aufgabe 3.3:

Ergänzen Sie in der Funktionssammlung »com_pc.c« eine Funktion mit der Header-Definition

```
void sendString(uint8_t *strg);
```

so dass das nachfolgende Hauptprogramm:

```
#include <avr/io.h>
#include "com_pc.h"
int main(){
  com_pc_init();
  sendString("Das ist ein Text.");
}
```

den Text »Das ist ein Text.« an den PC schickt.

Aufgabe 3.4:

- Lassen Sie den Modultest für jedes Testbeispiel auf Folie 41 die Übertragungsdauer wie in dem Programm auf Folie 24 bestimmen und mit ausgeben.
- Zeichnen Sie mit dem Logikanalysator die Signalverläufe von RxD und TxD für das erste Testbeispiel auf. Bestimmen Sie daraus die übertragenen Bit- und Bytefolgen und vergleichen Sie diese mit den Ausgaben des Python-Programms.

Aufgabe 3.5:

- Schreiben Sie ein Programm, das in einer Endlosschleife immer auf zwei Bytes wartet, diese nach der Vorschrift

  ```
  int16_t wert = (int16_t)(b1<<8|b2);
  ```

  (b1, b2 – erstes bzw. zweites empfangenes Byte) zu einer 16-Bit vorzeichenbehafteten Zahl zusammenfasst, diese negiert und verdoppelt und das Ergebnis zurücksendet
- Testen Sie das Programm mit der Eingabe 0x45A im HTerm.
- Programmieren Sie in Python einen Testrahmen, der zehn zufällig zu wählende Testbeispiele abarbeitet und für jedes Testbeispiel den Eingabewert, den Ausgabewert sowie den logischen Vergleichswert von Ist- und Sollausgabe ausdruckt.

Aufgabe 3.6:

- Schreiben Sie ein Programm, das zwei 2-Byte vorzeichenbehaftete Faktoren empfängt, multipliziert und ein 4-Byte-Produkt zurücksendet.
- Schreiben Sie ein Python-Programm, dass dieses Programm mit zehn zufällig ausgewählten Beispielen testet.

Aufgabe 3.7: USB + Bluetooth

- Stecken Sie das Bluetooth-Modul wie auf Folie 45 an JD (USART0) und den USB-Bluetooth-Dongle in den PC.
- Stellen Sie in der beschriebenen Weise eine Bluetooth-Verbindung her.
- Ändern Sie die Protokollparameter im PC-Programm und im HTerm auf 8 Datenbits, 1 Stoppbit, keine Parität, 9600 Baud und öffnen Sie die Verbindung mit dem COM-Port der Bluetooth-Verbindung.
- Führen Sie änliche Tests wie für die drahtgebundenen Kommunikation durch (incl. Messung Übertragungsdauer).

Der PC richtet für jedes Modul, das einmal angesteckt war, eine neue COM-Schnittstelle ein, über die das PC-Programm dann mit dem Mikrorechner kommuniziert.

Aufgabe 3.8: Modultest Vorzeichenzahlen

Ändern Sie das Mikrorechnerprogramm Folie 39 und das Python-Programm Folie 41 so, dass vorzeichenbehaftete Zahlen addiert, subtrahiert, dividiert und multipliziert werden.
Testbeispiele für das Python-Programm:

```
i_tuble = ((-23,45), (-89,-7), (0x7F, -17), (-58, -99))
```

- Welche Rechenergebnisse werden falsch berechnet?
- Wodurch werden die falschen Ergebnisse verursacht?

Hinweise: Im C-Programm müssen nur die Typen geändert werden. Im Pythonprogramm sind allen negativen 8-Bit-Werte vor der Umwandlung mit ord() durch Addition von 256 in die vorzeichenfreie Zahl mit gleichem Bytewert umzuwandeln. Nach dem Empfang und der Umwandlung mit chr() in eine Zahl ist von allen Werte ≥ 128 eine 256 abzuziehen. Von 16-Bit-Empfangswerten ist analog dazu 2^{16} abzuziehen.