Informatikwerkstatt, Foliensatz 4 PC-Kommunikation

G. Kemnitz

1. Dezember 2020

Inhalt:

Inhaltsverzeichnis

Interaktive Übungen:

1. Echoprogramm (echo)
2. Modularisierung und Modultest (com_pc)

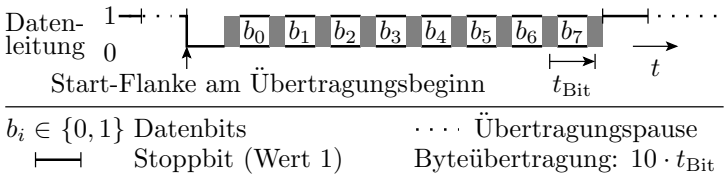
Anmerkungen zum Montagstest:

1. Wichtig für unseren Kurs: funktionsfähiges Programm Aufgabe 1.
2. Nützlich: Rechnen mit Binärzahlen auch bei WB-Überschreitung.
3. Gut zu können: Umgang mit Zeigern und Referenzen[1].

1 PC-Kommunikation

Kommunikationsprotokoll

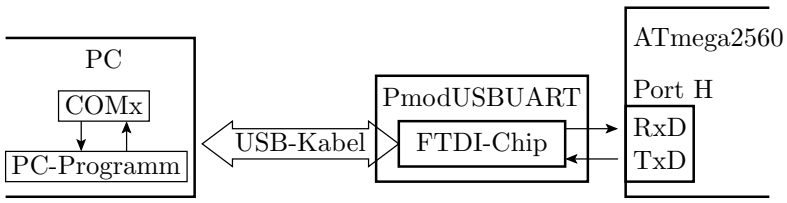
Der Datenaustausch zwischen Rechnern erfolgt seriell[2] über USB, Ethernet, CAN-Bus, ... Unsere PC-Kommunikation nutzt USART2, Kommunikationsprotokoll[3] 8N1[4], 9600 Baud:



- Bitzeit 1/9600s. (bis ca. 1000 Datenbytes pro s).

[1] Die Aufgabe war zu schlecht gestellt, um den Wissenstand dazu einzuschätzen.
[2] Seriell: Hintereinander über eine, statt parallel über viele Leitungen.
[3] Kommunikationsprotokoll: Vereinbarung, nach der der Datenaustausch zwischen zwei oder mehr Teilnehmern erfolgt.
[4] Format 8N1: 8 Datenbits, kein Paritätsbit und 1 Stoppbit.

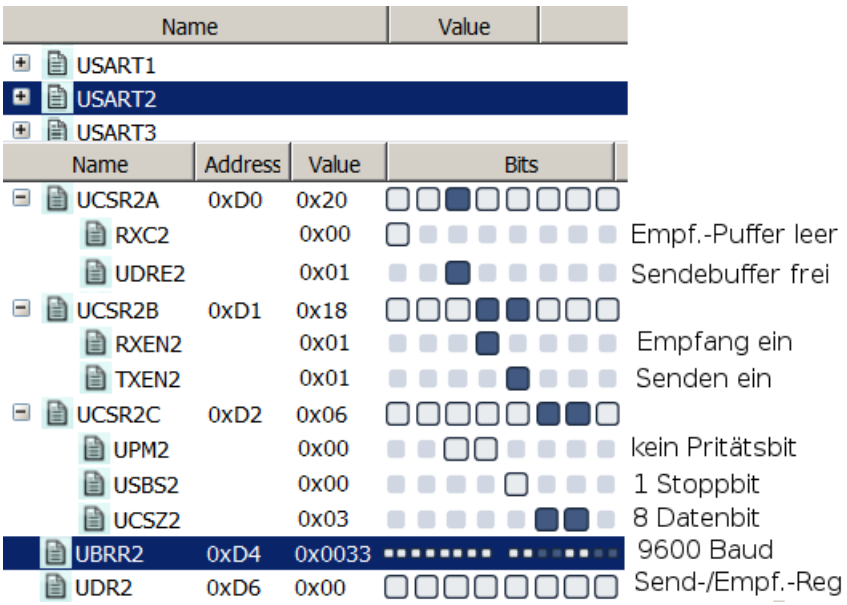
Kommunikationsfluss



- Der Mikrorechner kann zeitgleich je ein Byte 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 auf dem PC eine eigene COM-Port-Nummer.
- Das PC-Programm wird entweder HTerm oder ein selbst zu schreibendes Python-Programm sein.

Byte-Empfang und Senden im Mikrorechner

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



2 Echoprogramm

Echoprogramm (echo.c aus P04\F4-echo)

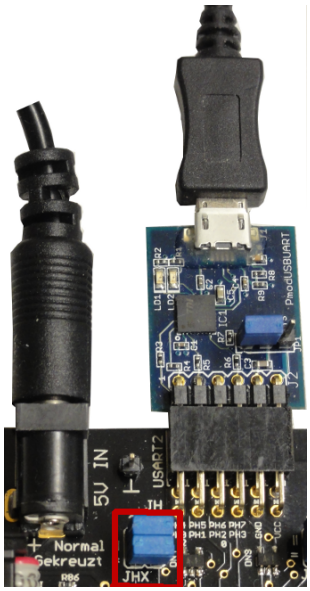
- Wiederhole: warte auf Bytes, zähle sie und sende sie zurück.

```
uint8_t daten;                  //Datei: echo.c
int main(void){
 UCSR2C=0b110;                  //Format 8N1
 UBRR2=51;                      //9600 Baud
 UCSR2B=(1<<RXEN2)|(1<<TXEN2);//Empf. + Sender ein
 DDRJ = 0xFF;                   //LEDs als Ausgänge
 PORTJ= 0;                      //LED-Ausgabe 0x00
 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++;                      //LED-Ausgabe erhöhen
 }
}
```

Test des Echo-Programms

Hardware vorbereiten:

- Spannung ausschalten.
- Jumper JHX »gekreuzt (=)«.
- PModUSBUSART Kontrolle,
  Jumper wie im Bild, und
  und an JH oben stecken.
- PModUSBUSART mit
  PC verbinden.
  Spannung einschalten.

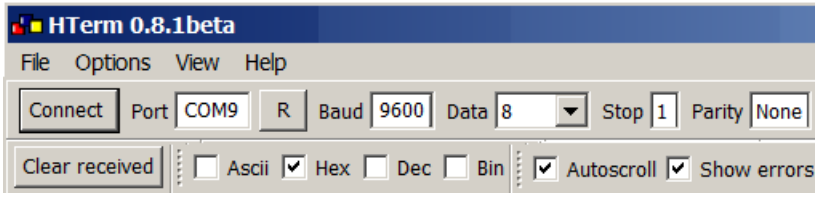


Software vorbereiten:

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

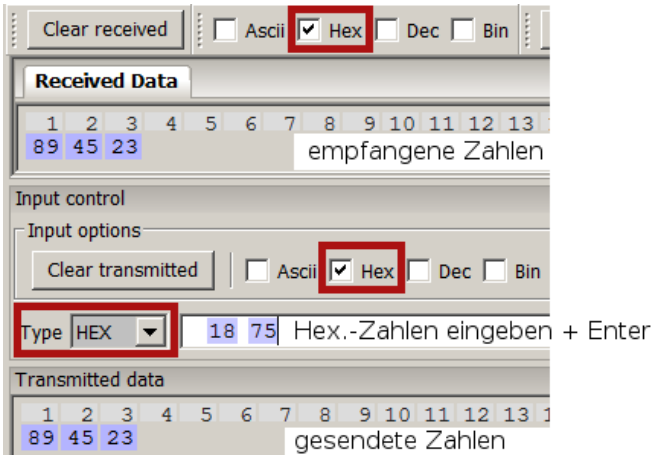
Verbindung mit HTerm herstellen

- Auf dem PC HTerm starten.
- COM-Port auswählen[5].
- 9600 Baud, 8 Daten-, 1 Stopp- und kein Paritätsbit einstellen.
- Verbindung herstellen (Connect).



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

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



- Alle versendeten Zahlen werden zurückgesendet.

3 Textdarstellung

Zeichendarstellung im ASCII-Code

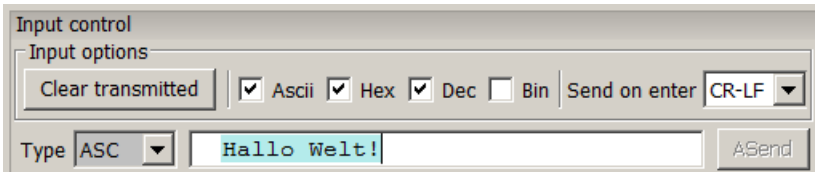
Buchstaben, Ziffern und andere Zeichen werden als Bytes und Texte als Felder von Bytes dargestellt. ASCII-Code:

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

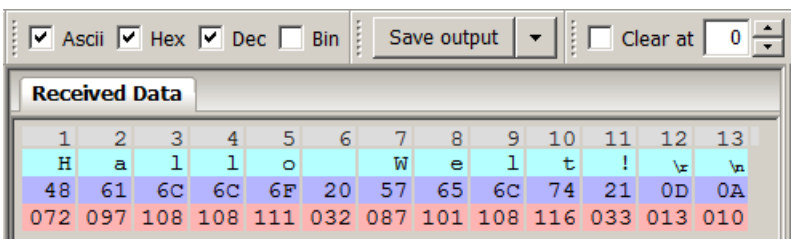
LF – Zeilenvorschub; CR – Wagenrücklauf; ␣ – Leerzeichen

Senden und Empfang von Texten

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



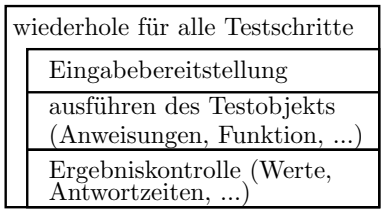
Empfangene Daten als Zeichen- und Zahlenfolge:



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

4 Modultest vom PC aus

Testrahmen



Die Basisfunktionen für den Test vom PC:

- Übergabe von Eingabe-Bytes und die
- Rückgabe von Ergebnis-Bytes zur Auswertung.

sind im Echo-Programm enthalten.

Modularisierung des Echoprogramms
Aufteilung des Echoprogramms »echo.c« von Folie 3 in nachnutzbare Module für den Test von Programmbeisteinen:

```
int main(void){
// ---------- Initialisierung ----------------------
 UCSR2C=0b110;                // Format 8N1
 UBRR2=51;                    // 9600 Baud
 UCSR2B=(1<<RXEN2)|(1<<TXEN2);// Empf. + Sender ein
// -------------------------------------------------
 while(1){
// ---------- Empfangen eines Bytes ---------------
  while (!(UCSR2A & (1<<RXC2)));//warte auf Byte
  daten = UDR2;               //Byte übernehmen
// ---------- Versenden eines Bytes ---------------
  while (!(UCSR2A &(1<<UDRE2)));//warte Puffer frei
  UDR2 = daten;               //Byte übergeben
} // ------------------------------------------------
}
```

Funktionen für die PC-Kommunikation (com_pc.c)

```
#include <avr/io.h>
//Initialisierung von USART2 (8N1, 9600 Baud)
void com_pc_init(){
 UCSR2C=0b110;                 //Format 8N1
 UBRR2=51;                     //9600 Baud
 UCSR2B=(1<<RXEN2)|(1<<TXEN2); //E+S 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 »com_pc.h« für den Export

```
#ifndef COM_PC_H_
#define COM_PC_H_

#include <avr/io.h>

 void com_pc_init();          //Init. USART2
 uint8_t getByte();           //Byte empfangen
 void sendByte(uint8_t dat); //Byte versenden

#endif /* COM_PC_H_ */
```

- Enthält die drei Funktionsdefinitionen ohne Anweisungen.
- #...: Precompiler-Anweisungen. Ausführung (Textverarbeitung) vor dem Compilieren.
- #ifndef ... #define ... #endif verhindern, dass Definitionen mehrfach in den zu übersetzenden Quelltext übernommen werden.

Testrahmen (Hauptprogramm)

```
#include <avr/io.h>   // Anmerkung *1
#include "com_pc.h"
uint8_t d;
int main(void){
 com_pc_init();       //Init. USART2
 while(1){            //Endlosschleife
  d = getByte();      //Byte empfangen
  sendByte(d);        //Byte zurücksenden
 }
}
```

*1: überflüssig, da »avr/io.h« in »com_pc.h« eingefügt wird, falls es vorher noch nicht eingefügt wurde.

Modultest vom PC – Ein Testobjekt

Testobjekt sei folgende Berechnungssequenz:

```
uint8_t a, b, s, d, q, r;
uint16_t p;
...
s = a + b;    // Summe
d = a - b;    // Differenz
p = a * b;    // Produkt
q = a/b;      // ganzzahliger Quotient
r = a%b;      // Divisionsrest
```

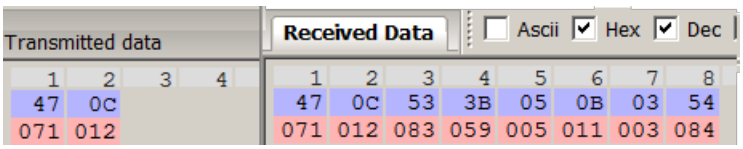
Darum soll ein Rahmenprogramm gelegt werden, das

- in einer Endlosschleife
- vom PC auf zwei Bytes für a und b wartet,
- die zu testenden Anweisungen ausführt und
- 8 Bytes (s, d, 2×p, q und r) zurückschickt.

```
#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();
  //-- zu testende Anweisungen -----
  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 »F4-com_pc« öffnen.
- »Dragon« und Compiler-Optimierung -O0 auswählen.
- Übersetzen. Debugger starten. Programm freilaufend starten.
- 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	$3 \cdot 2^8 + 84 = 852$

5 Aufgaben

Aufgabe 4.1: HA bei weniger als 7 P im Test[6]

1. Ergänzen Sie im Kommentar den zugewiesenen Wert in Hexadezimaldarstellung und ab Zeile 3 auch die Dezimalwerte:

```
1 int8_t a, v1 = 35;   //
2 uint8_t b, v2 = 60;  //
3 a = v1 | 20;         //
4 a = v1 << 3;         //
5 b = v2 & (1 << 2);   //
6 b = v2 << 4;         //
7 b = v1 - 40;         //
```

2. Vervollständigen Sie Zeile 2 so, dass den Bits 0 bis 3 von a die Bits 4 bis 7 von b und den Bits 4 bis 7 von a die negierten Bitwerte von c Bit 2 bis Bit 5 zugewiesen werden.

```
1 uint8_t a, b = 25, c = 37;
2 a =
```

3. Welcher Wert steht nach Ausführung mit den Beispielzahlen in a?

Aufgabe 4.2: Abarbeitung der Interaktiven Übungen

1. Ausprobieren Echo-Programm (ab Seite 3).
2. Ausprobieren Modultest (ab Seite 7).

Aufgabe 4.3: Textdarstellung und -ausgabe

Durch welche Zahlenfolge wird der nachfolgende Text dargestellt:

```
"Informatikwerkstatt , Uebung3"
```

1. Lösen Sie die Aufgabe mit der ASCII-Tabelle auf Folie 5.
2. Kontrollieren Sie das Ergebnis, in dem Sie die Zeichenkette mit dem HTerm versenden und zusätzlich als ASCII-Folge anzeigen lassen.
3. Kontrollieren Sie das Ergebnis mit folgendem Programm:

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

[6] Abgabe ha-iw@in.tu-clausthal.de bis Mo. 16.11.20.

Aufgabe 4.4: Wiederhole bis zum »Nullbyte«

Wenn C wie in der nachfolgenden Programmzeile

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

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

1. Kontrollieren Sie das mit dem Debugger.

2. Schreiben Sie das Programm aus der vorherigen Aufgabe so um, dass es nicht genau 28 Zeichen ausgibt, sondern alle Zeichen bis vor dem Zeichenwert null.

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

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

Aufgabe 4.5: Zeitmessung Warteschleife

1. Schreiben Sie ein c-Programm, dass auf ein Byte vom PC wartet, die folgende Warteschleife

   ```
   uint32_t ct;
   ...
   for (ct=0; ct<500000; ct++);
   ```

   abarbeitet und das Byte zurücksendet.

2. Testen Sie das Programm mit HTerm und schätzen Sie die Zeit vom Versenden bis zum Empfang. (Man kann in HTerm empfangene Daten mit Zeitstempel aufzeichnen.)

Aufgabe 4.6: Textausgabe

Ergänzen Sie in der Funktionssammlung »com_pc.c« eine Funktion zur Textausgabe mit der Aufrufschnittstelle:

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

Als Testbeispiel soll 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 schicken.

Aufgabe 4.7: Modultest vom PC aus

1. 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.

2. Testen Sie das Programm mit der Eingabefolge (0x04, 0x5A) im HTerm.