Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Fakultät: Technik und Informatik || Department: Information und Elektrotechnik

Laborbericht

MPP-4 Serielle Schnittstelle

Labor Prozessortechnik

Professor: **Prof. Dr.-Ing. K.-R. Riemschneider**

Semestergruppe: E4
Datum des Labors: 07.November 2016

Protokollant: Nils Parche
Matrikelnummer: 2210363
E-Mail: nils.parche@haw-hamburg.de

Assistent: **Marvin Janz**
Matrikelnummer: 2245023
E-Mail: marvin.janz@haw-hamburg.de

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Inhaltsverzeichnis

1 Einleitung

Das letzte Labor in Mikroprozessortechnik beschäftigt sich mit der seriellen Übertragung. Es sollte eine Verbindung des Mikroprozessors mit dem PC über ein Hyper Terminal und einer RS 232 Schnittstelle erstellt werden. Mithilfe der erstellten Verbindung wurden anschließend drei verschiedene Übertragungsprozesse gemäß Aufgabenstellung realisiert.
Diese waren:

1. Wiederholte Ausgabe eines Zeichens auf dem Hyper Terminal in Abhängigkeit:
   - der Bitrate in $\frac{Bits}{sek}$
   - der Anzahl der Datenbits
   - der Paritätsgenerierung
   - der Anzahl der Stoppbits
2. Ausgabe eines vorgegebenen Textes auf dem Hyper Terminal vom Mikrocontroller
3. Übertragung eines frei wählbaren Textes vom Hyper Terminal zum Mikrocontroller unter der Beachtung, dass:
   - Kein Zeichenüberlauf stattfindet
   - Schreibfehler korrigiert werden können
   - Eine Echo Funktion aktiviert, oder deaktiviert werden kann

2 Aufgabenteil 1 Wiederholte Ausgabe eines Zeichens

Für die wiederholte Ausgabe eines Zeichens wurden Übertragungen mit vier verschiedenen Einstellungen getätigt. Für jede Übertragung wurde jeweils das große ***T*** verwendet.

Tabelle 1: Übersicht der Einstellungen zur Übertragung eines Zeichens an das Hyper Terminal

Variante	Datenrate in Baud	Datenbits	Parität[1]	Stopbits
1	115200	8	no parity	1
2	115200	8	even	2
3	19200	8	odd	1
4	2400	7	no parity	1

Für die Umsetzung der Baud Rate in dem Mikrocontroller musste das UART Integer Baud Rate Divisor Register gesetzt werden. Die Einstellung der richtigen Baud Rate wurde mit der Formel

$$BRD = \frac{SysClk}{16 \cdot gewuenschte\ Bitrate} \tag{1}$$

[1] berechnet.
Das Ergebnis ist anschließend eine Bruchzahl. Diese wird in ihren ganzzahligen (integer) Teil und in ihren gebrochenen (fractional) Teil unterschieden.
Der integer Teil wird in das

UARTx_IBRD_R [2]

Register geschrieben.

Der gebrochene Teil wird in das

UARTx_FBRD_R [2]

Register geschrieben.
Für alle im Labor vorgenommenen Übertragungen wurde der UART6 verwendet. Des weiteren war der SysClk für die Berechnung der Baud Rate 25MHz. Dies musste so gewählt werden da für die Übertragung der Hauptoszillator des Prozessors initialisiert wurde.

[1] odd steht für ungerade Parität, even für gerade Parität

2.1 Überlegung zur Ausgabe des übertragenen Signales

Während des Laborversuches war es angedacht, wie auch im Kapitel 2.4 dargestellt, das übertragene Signal mit einem Oszilloskope zu analysieren. Eine im Vorfeld getätigte Überlegung wie der Signalverlauf von Beginn bis zum Ende der Übertragung aussehen würde. Diese wird im Folgenden in nummerisch aufsteigender Reihenfolge für das TTL Signal aufgelistet:

1. Startbit: Der erste erscheinende Spannungssprung wird durch das Startbit entstehen. Dies kennzeichnet den Beginn der Übertragung.

2. Datenbits: Direkt nach dem Startbit beginnt die eigentliche Übertragung des Signales. Hierbei muss auf die gewählte Länge des Datenbits geachtet werden. Handelt es sich um ein Zeichen aus dem ACSII-Code sind 7 Bits für den einfachen, sowie 8 Bits für den erweiterten Code (mit Umlauten/Sonderzeichen) von nöten. Andere Übertragungslängen erfordern eine sehr gute Abstimmung von Sender und Empfänger.
   Wichtig bei der Übertragung der Datenbits: LSB first!

3. Paritätbit: Nach der Datenübertragung wird bei aktivierter Parität ein weiteres Bit übertragen. Je nach Einstellung auf odd oder even ist dies eins oder null.

4. Stopbit: Als letztes wird ein Stopbit übertragen, welches die Beendigung der Datenübertragung kennzeichnet.

Zu beachten ist auch, dass das RS232 Signal invertierend zur eigentlichen binären Logik ist. Dies bedeutet, dass eine logische 1 einen Spannungsabfall auf $\approx -10V$ hervorruft, während eine 0 einen Spannungsanstieg auf $\approx +10V$ verursacht.

2.2 Programmalgorithmus für Variante 1

Da sich die vier Varianten für die Übertragung sich sehr stark ähneln, wurde davon abgesehen, diese jeweils aufzulisten.
Im Folgenden wird die Variante 1 der Tabelle 2 aus dem Kapitel 2 komplett dargestellt:

```
1  /*********************************************************************
2        Autor: NPA, MJA
3        Datum: 19.12.2016
4        Dateiname: transmit
5        Version: 1.0
6   *********************************************************************/
7
8  /*
9   * Baudrate - Calculation
10  *
11  * BRD = SysClk / (16 * gewuenschte Bitrate)
12  * IBRD = int(BRD)
13  * FBRD = int( (BRD-IBRD * 64 + 0.5)
14  */
15
16 /*
17  *    PP0 -> RxD6
18  *    PP1 -> TxD6
19  */
20
21 /*
22  * VARIANTE 1:
23  * 115200 bit/s
24  * 1 stop bit
25  * no parity
26  */
27
28 #include <stdio.h>
29 #include <stdint.h>
30 #include "tm4c1294ncpdt.h"
31
32 #define BITRATE 115200
33 #define SYSCLK 25000000.0
34 #define BRD SYSCLK / (16 * BITRATE)
35 #define IBRD (int) BRD
36 #define FBRD (int) (((BRD-IBRD) * 64) + 0.5 )
37 #define BITDATA 0x54
38 #define WAITLOOPLENGTH 20000
39
40 void main(void) {
41
42       uint8_t wt=0;
43       uint32_t i=0;
44
45       /*****************************
46        * CLOCK - Settings
47        *****************************/
48       // switch over to main quartz oscillator at 25MHz
49       // clear MOSC power down, high oscillator range setting,
50       // and no crystal present setting
51       SYSCTL_MOSCCTL_R &= ~(SYSCTL_MOSCCTL_OSCRNG | SYSCTL_MOSCCTL_PWRDN
52       | SYSCTL_MOSCCTL_NOXTAL);
53       // increase the drive strength for MOSC
54       SYSCTL_MOSCCTL_R |= SYSCTL_MOSCCTL_OSCRNG;
55       // set the main oscillator as main clock
56       SYSCTL_RSCLKCFG_R = SYSCTL_RSCLKCFG_OSCSRC_MOSC;
57
58       /*****************************
```

```
59          * PORTC - Define
60          *****************************/
61         SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R2; wt++;
62         GPIO_PORTC_AHB_DEN_R = 0x30;
63         GPIO_PORTC_AHB_DIR_R = 0x30;
64
65         /*****************************
66          * PORTP - Define
67          *****************************/
68         SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R13; wt++;
69         GPIO_PORTP_DEN_R |= 0x02;
70         GPIO_PORTP_DIR_R |= 0x02;
71         // alternative pin function PP1 -> TxD6
72         GPIO_PORTP_AFSEL_R |= 0x02;
73         // controls the MUX for a pin example PP.1 -> TxD6
74         // each nipple (4-Bit) configure Output Bit
75         GPIO_PORTP_PCTL_R |= 0x00000010;
76
77         /*****************************
78          * UART6 - Define
79          *****************************/
80         // enable clock at UART module 6
81         SYSCTL_RCGCUART_R |= SYSCTL_RCGCUART_R6; wt++;
82         // disable UART during initialization
83         UART6_CTL_R &= ~UART_CTL_UARTEN;
84         // configure BAUD rate
85         // set DIVINT of BRD
86         UART6_IBRD_R = IBRD;
87         // set DIVFRAC of BRD
88         UART6_FBRD_R = FBRD;
89         // set Frame-Parameter
90         // WLEN_x = bits
91         // PEN 0 parity disable 1 enable
92         // EPS 0 odd parity 1 even
93         // STP2 0 one stop bit 1 two stop bits
94         UART6_LCRH_R |= (UART_LCRH_WLEN_8);
95         // after initialization re-enable UART0
96         UART6_CTL_R |= (UART_CTL_UARTEN | UART_CTL_TXE); wt++;
97
98         while(1) {
99               GPIO_PORTC_AHB_DATA_R = 0x20;
100              while(!(UART6_FR_R & UART_FR_TXFE));
101              UART6_DR_R = BITDATA;
102              GPIO_PORTC_AHB_DATA_R = 0x00;
103              for(i=0; i<WAITLOOPLENGTH; i++);
104        }
105  }
```

Listing 1: Programmcode Übertragen eines Zeichens

2.3 Parametereinstellung für Variante 2 , 3 , 4

Für die übrigen Varianten wurden folgende Parameter aus dem Ablauf in Kapitel 2.2 verändert.

Zeile 32: Bitrate mit # define BITRATE auf die gewählten Bits pro Sekunde gestellt

Zeile 94:

- Veränderung der Bitlänge mit dem Befehl $UART\_LCRH\_WLEN\_x$
- Initialisierung der Paritätsgenerierung mit dem Befehl $UART\_LCRH\_PEN$
- Parität mit dem Befehl $UART\_LCRH\_EPS$ auf even gesetzt
- Parität durch das **Auslassen** des Befehles $UART\_LCRH\_EPS$ auf odd gesetzt
- Anzahl der Stopbits mit dem Befehl $UART\_LCRH\_STPx$ auf 1 oder 2 (anstelle von x) gesetzt
- Kein Stoppbit durch auslassen des Befehles $UART\_LCRH\_STP$

2.4 Auflistung der gemessenen Signale

Im Folgenden werden die im Labor festgehaltenen Messungen, welche über das Oszilloskop festgehalten wurden aufgelistet.

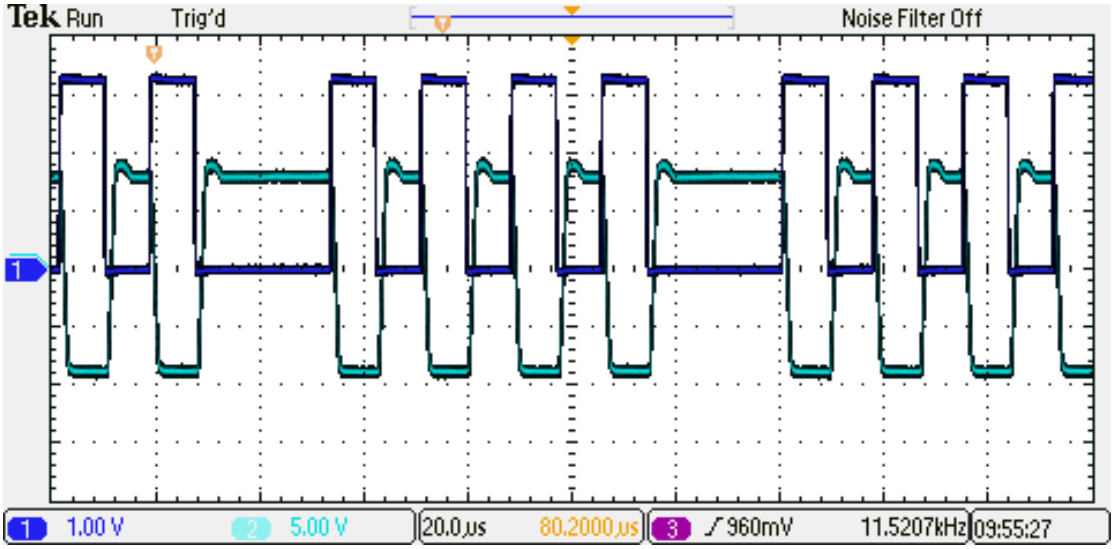

Abbildung 1: Serielle Übertragnung des Zeichens T mit der Einstellung der Variante 1

Abbildung 1. Zu erkennen sind die beiden vom Typ TTL und RS 232. Das RS232 Signal verliert aufgrund der hohen Übertragungsrate an Qualität in der Anstiegszeit , sowie der Genauigkeit der Endwertspannung (es entsteht ein Überschwinger).

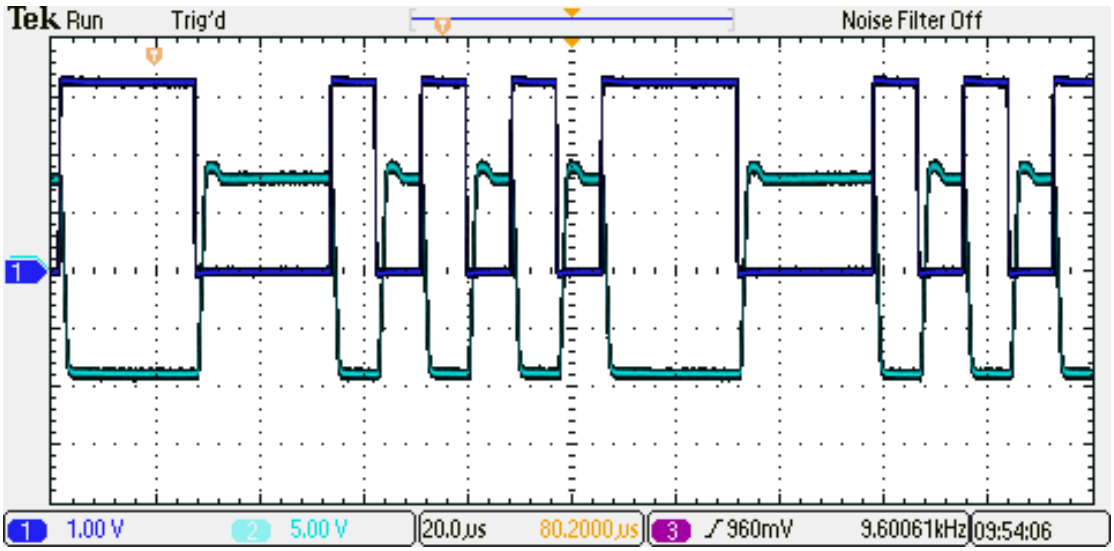

Abbildung 2: Serielle Übertragnung des Zeichens T mit der Einstellung der Variante 2

Abbildung 2. Erneut ist , wie in Abbildung 1 aufgrund der hohen Baudrate Ein Qualitätsverlust im Bereich der slew rate , sowie ein Überschwinger zu erkennen.

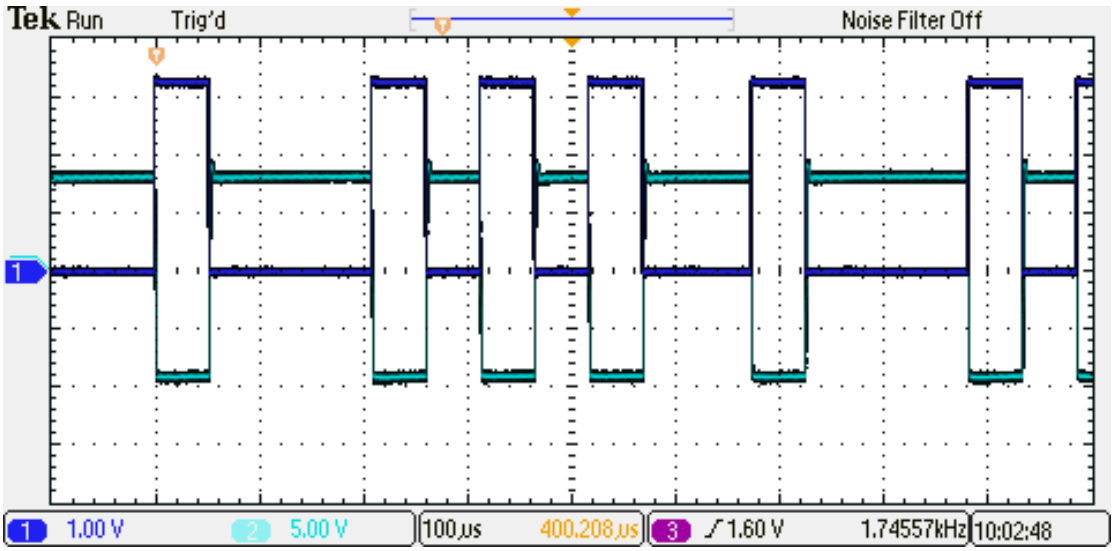

Abbildung 3: Serielle Übertragung des Zeichens T mit der Einstellung der Variante 3

Abbildung 3. Durch die verminderte Datenübertragungsrate ist die slew rate des RS 232 Signales verringert. Es ist allerdings weiterhin ein Überschwinger wie in Abb. 1, sowie Abb. 2 zu beobachten.

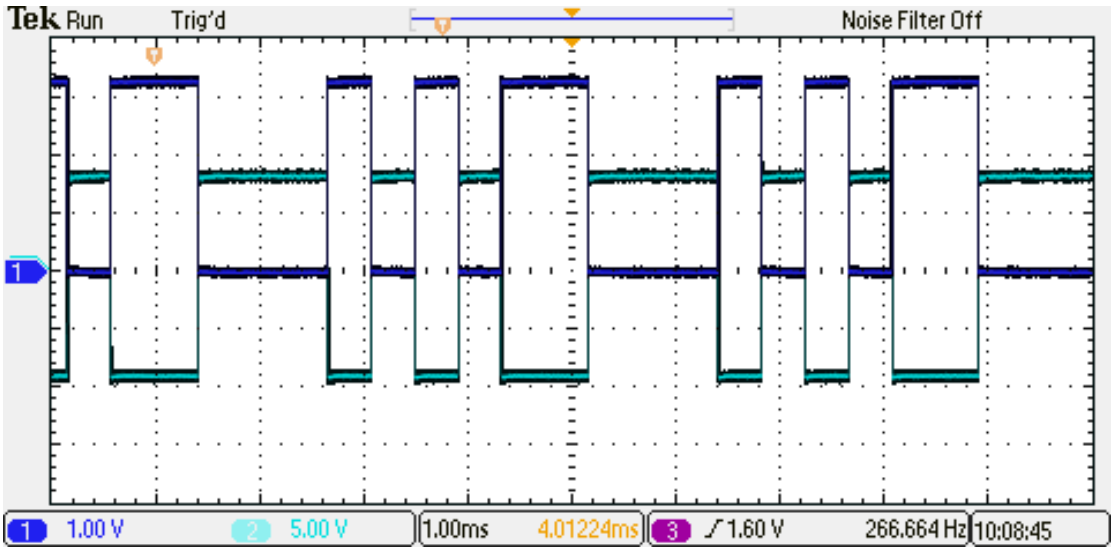

Abbildung 4: Serielle Übertragnung des Zeichens T mit der Einstellung der Variante 4

Abbildung 4. Während des Versuches wurde bei dieser die geringste Baud-Rate eingestellt. Dies hat zur folge, dass die Qualität des RS 232 Signales von allen vier Varianten die beste darstellt. Es ist weder eine slew rate, noch ein Überschwinger des Signales zu beobachten.
Zu beachten ist allerdings, dass bei einer verminderten Baud Rate die Anzahl der maximal zu übertragenen Zeichen ebenfalls verringert wird!
Dieser Aspekt war für das vierte Labor nicht verlangt.

2.4.1 Handschriftliche Zeichnung der Signalverläufe

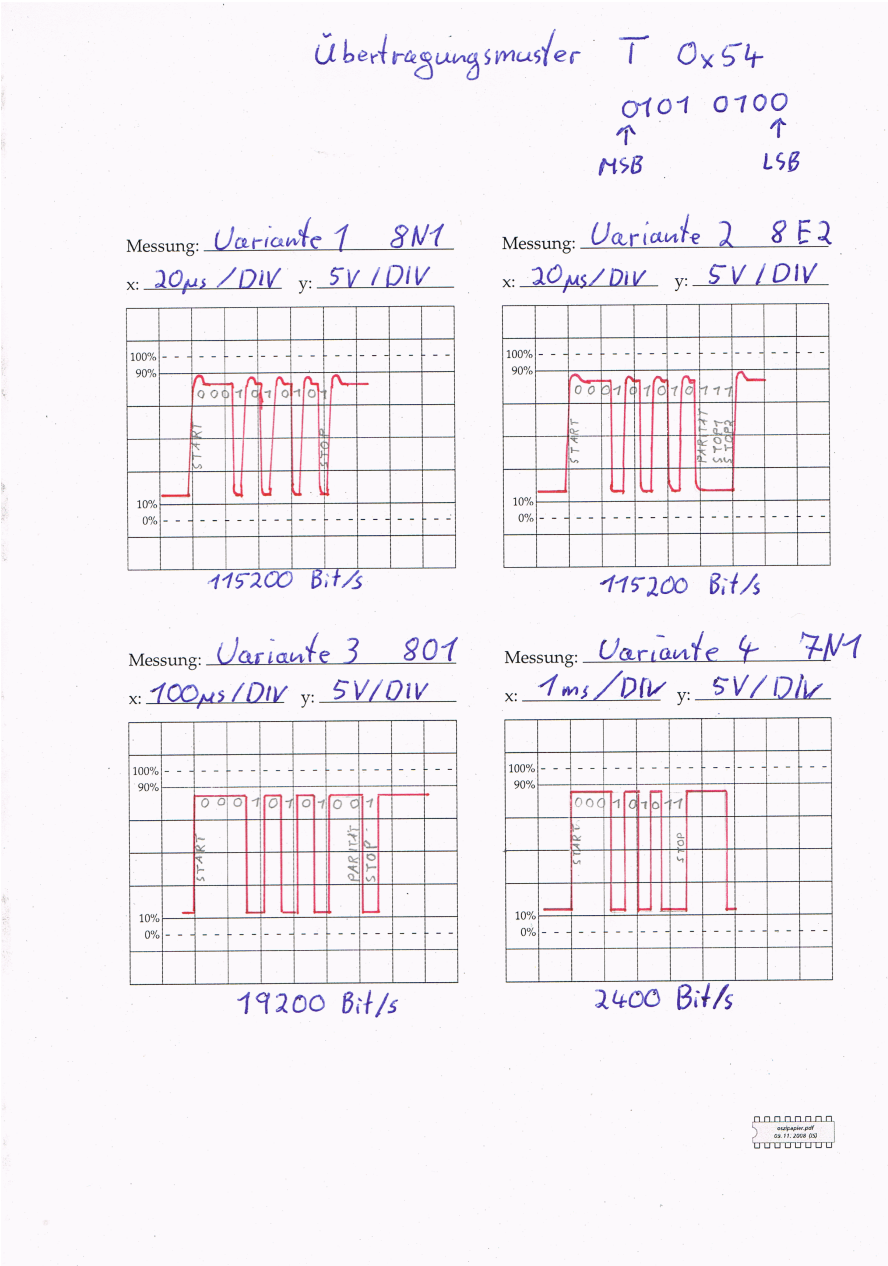

Abbildung 5: Handschriftliche Zeichnung der Signalverläufe

2.5 Fazit

Bei der Übertragung ist es wichtig, einen klar und verständlichen Programmablauf zu programmieren, da die Grundeinstellung des Controllers immer die gleiche ist. Werden die fest einzustellenden Werte klar strukturiert, ist es anschließend nicht mehr kompliziert, unterschiedliche Übertragungsparameter einzustellen. Während der Messungen war zu erkennen, dass das eigentliche Serielle Ausgangssignal in seiner Qualität immer gleich blieb, während sich die Umsetzung auf ein RS 232 Signal mit der Baud Rate veränderte. War diese hoch, litt die Umsetzung an Überschwingern und es konnte eine Anstiegs-, sowie Abfallzeit der Flanken beobachtet werden.

3 Aufgabenteil 2 Ausgabe einer Zeichenfolge

3.1 Beschreibung und Lösungsansatz

Der Aufgabenteil 2 verlangte es, einen vorgegebenen Text auf der Konsole des Hyper Terminals auszugeben. Der zu übertragenen Text lautete:

Praktikum Serielle Uebertragung vom 19.12.2016.

Versuchsteilnehmer:

Nils Parche
Marvin Janz

Die Konfiguration des Controllers wurde dabei aus der ersten Aufgabe übernommen. Die vorgenommen Einstellungen waren:

- Für die Baudrate: $9600\frac{Bit}{s}$
- Für den System Clock 25 MHz (Hauptoszillator)
- UART6 für die serielle Übertragung gewählt
- Konfiguration des $UART6\_LCRH\_R$ Registers: [2]
  - $UART\_LCRH\_WLEN\_7$ Übertragen von 7 Datenbits
  - $UART\_LCRH\_PEN$ Initialisieren der Paritäts Abfrage
  - $UART\_LCRH\_EPS$ Einstellen gerader Parität
  - Kein Stoppbit durch auslassen des Befehles $UART\_LCRH\_STP$

Der zu übertragenen Text wurde, mit Formatierung, in ein char Array geschrieben und über eine for Schleife an das Hyper Terminal übergeben. Siehe dazu Programmcode Listing 2, Zeile 92-99.

3.2 Programmalgorithmus

Folgender Programmalgorithmus kam zum Einsatz:

```
/*********************************************************************
      Autor: NPA
      Datum: 13.12.2016
      Dateiname: Datenausgabe
      Version: 1.0
 *********************************************************************/
/*
 * Baudrate - Calculation
 * BRD = SysClk / (16 * gewuenschte Bitrate)
 * IBRD = int(BRD)
 * FBRD = int( (BRD-IBRD * 64 + 0.5)
 *    PA0 -> RxD0
 *    PA1 -> TxD0
 * 9600 bit/s
 * databits: 7
 * 1 stop bit
 * parity even
 */
#include <stdio.h>
#include <stdint.h>
#include "tm4c1294ncpdt.h"

#define BITRATE 9600
#define SYSCLK 25000000.0
#define BRD SYSCLK / (16 * BITRATE)
#define IBRD (int) BRD
#define FBRD (int) (((BRD-IBRD) * 64) + 0.5 )

#define BITDATA 0x54
#define WAITLOOPLENGTH 20000

void main(void) {

      uint8_t wt=0;
      uint32_t i=0;

      /*****************************
       * CLOCK - Settings
       *****************************/
      // switch over to main quarz oscillator at 25MHz
      // clear MOSC power down, high oscillator range setting,
      // and no crystal present setting
      SYSCTL_MOSCCTL_R &=
      ~(SYSCTL_MOSCCTL_OSCRNG | SYSCTL_MOSCCTL_PWRDN | SYSCTL_MOSCCTL_NOXTAL);
      // increase the drive strength for MOSC
      SYSCTL_MOSCCTL_R |= SYSCTL_MOSCCTL_OSCRNG;
      // set the main oscillator as main clock
      SYSCTL_RSCLKCFG_R = SYSCTL_RSCLKCFG_OSCSRC_MOSC;

      /*****************************
       * PORTC - Define
       *****************************/
      SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R2; wt++;
      GPIO_PORTC_AHB_DEN_R = 0x30;
      GPIO_PORTC_AHB_DIR_R = 0x30;

      /*****************************
       * PORTP - Define
       *****************************/
      SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R13; wt++;
```

```
61        GPIO_PORTP_DEN_R |= 0x02;
62        GPIO_PORTP_DIR_R |= 0x02;
63        // alternative pin function PA1 -> TxD6
64        GPIO_PORTP_AFSEL_R |= 0x02;
65        // controls the MUX for a pin example PP.1 -> TxD6
66        // each nipple (4-Bit) configure Output Bit
67        GPIO_PORTP_PCTL_R |= 0x00000010;
68
69        /*****************************
70         * UART - Define
71         *****************************/
72        // enable clock at UART module 6
73        SYSCTL_RCGCUART_R |= SYSCTL_RCGCUART_R6; wt++;
74        // disable UART during initialization
75        UART6_CTL_R &= ~UART_CTL_UARTEN;
76        // configure BAUD rate
77        // set DIVINT of BRD
78        UART6_IBRD_R = IBRD;
79        // set DIVFRAC of BRD
80        UART6_FBRD_R = FBRD;
81        // set Frame-Parameter
82        // WLEN_x = bits
83        // PEN 0 parity disable 1 enable
84        // EPS 0 odd parity 1 even
85        // STP2 0 one stop bit 1 two stop bits
86        UART6_LCRH_R |= (UART_LCRH_WLEN_7 | UART_LCRH_PEN | UART_LCRH_EPS);
87        // after initialization re-enable UART0
88        UART6_CTL_R |= (UART_CTL_UARTEN | UART_CTL_TXE); wt++;
89
90        char text[] = "\n\n\r Praktikum Serielle Uebertragung vom 19.12.2016.
91        \n\n\r Versuchsteilnehmer:
92        \n\n\r Nils Parche \n\r Marvin Janz";
93
94        for(i=0; i<sizeof(text); i++) {
95              UART6_DR_R = text[i]; //Transmit a single char
96              while(!(UART6_FR_R & UART_FR_TXFE)); // Wait until char has been transmitted
97        }
98        while(1) {} //Endless loop after transmitting
99  }
```

Listing 2: Programmcode Übertragen eines ausgewählten Textes

3.3 Struktogramm Aufgabe 2

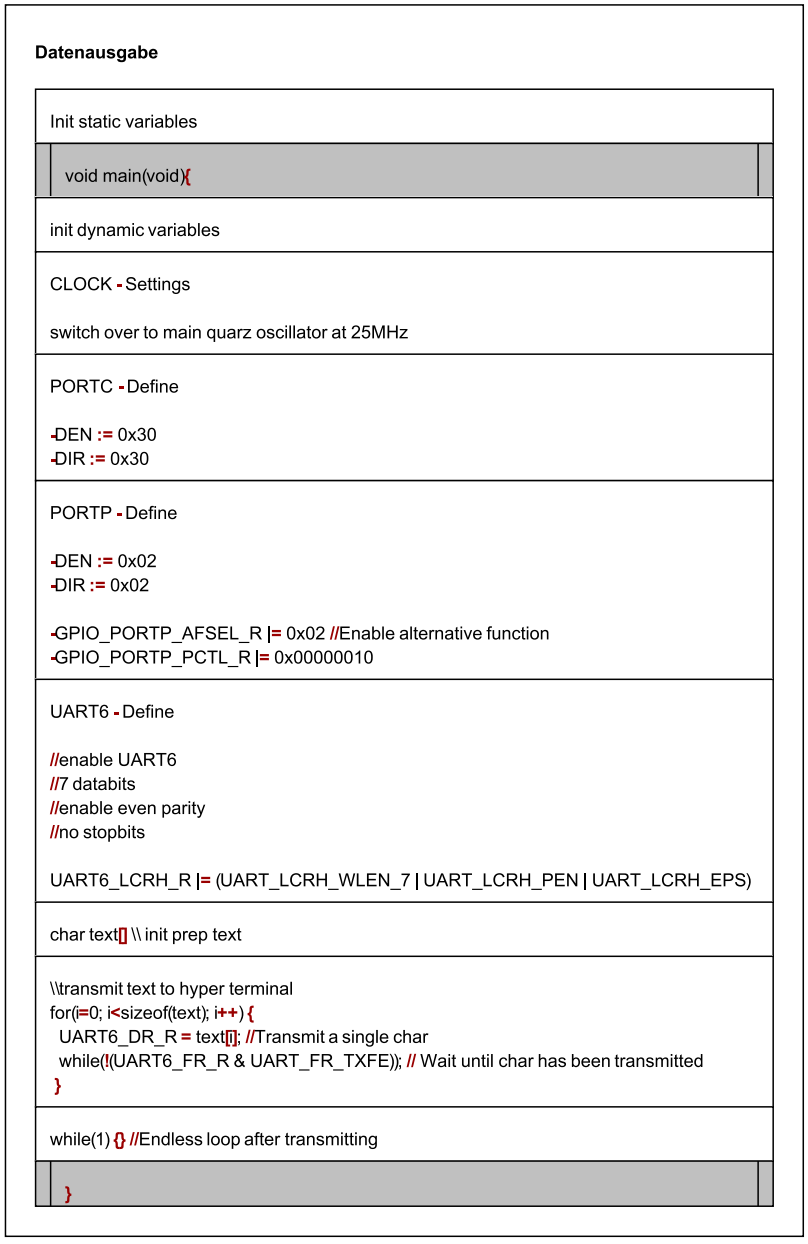

Abbildung 6: Nassi Schneidermann Diagramm zur Datenausgabe

3.4 Fazit

Die Übertragung eines vorgewählten Textes erwieß sich als nicht kompliziert. Durch die vorhergehenden Aufgaben konnte die Einstellung des Controllers erneut ohne große Komplikationen vorgenommen werden. Die Übertragung der Zeichenkette erforderte hingegen etwas Überlegung. Es ist mit der hier aufgeführten Lösung zum Beispiel schwierig eine dynamische Eingabe durch einen Benutzer zu übersetzten. Das Hauptproblem stellt hierbei die Formatierung durch z.B. Zeilenumbrüche dar.

4 Aufgabenteil 3 Empfangsprogramm

4.1 Beschreibung und Lösungsansatz

Es soll ein Programm entwickelt werden mit dem es möglich ist Zeichen über die Serielle Schnittstelle des Controllers einzulesen. Die Parameter der Seriellen-Schnittstellen entsprechen dem des vorherigen Aufgabenteils (9600 bit/s, 7E1). Um den Speicherbedarf, der mit der Anzahl zu speichernden Zeichen begrenzt wird, wurde ein Array der Größe 16 als Einlesespeicher festgelegt. Des weiteren sollen die nachfolgenden Punkte in den Programmablauf eingebunden werden.

- **Es sollen Schreibfehler mit dem Zeichen Backspace (0x08) korrigiert werden können:**
  Jedes eingelesene Zeichen wird als ASCII-Code repräsentiert. Die Backspace-Taste besitzt den Code 0x08 und somit kann dieser abgefragt werden. Im Programmablauf wird bei erkennen der Backspace-Taste das vorherige Zeichen gelöscht. Zusätzlich wird die Position des Cursor um eine Stelle nach vorne versetzt, es sei denn der Cursor befindet sich bereit am Anfang.

- **Die Eingabe kann beendet werden mit „End of file“ Tastatureingabe (Ctrl + d):**
  Durch die Beendigung der Eingabe (Tastatur Ctrl+d) wird im Programmablauf das einlesen abgebrochen und der aktuelle Lesespeicher zur Kontrolle an den Terminal zurückgesendet.

- **Mit einem externen Schalter kann zwischen non-Echo- und Echo-Modus gewählt werden:**
  Bei aktivem Echo-Modus wird jedes Empfangendes Zeichen sofort an den Terminal zurück gesendet. Es dient der Visualisierung der Übermittelten Zeichen.

4.2 Programmalgorithmus

Das Programm wurde so angelegt, dass nur maximal 15 Zeichen in einem Speicher geladen werden können. Durch diese Randbedingung ist eine bedingte Schleife erforderlich, die genau 15 Werte einließt oder mit der Tastatureingabe „End of file“ beendet wird. Sollte nach dem 15 Wert weitere Zeichen übermittelt werden, wird eine Fehlermeldung mit den aktuellen Lesespeicher an den Terminal übermittelt. Erst nachfolgend wird das neu empfangende Zeichen in den Speicher geladen.
Mit der Backspace-Taste kann die Eingabe der zuvor getätigten Zeichen gelöscht werden. Im Programmablauf wird bei jeder erkannten BS-Taste der Inhalt des Empfangsspeichers an der vorherigen [i-1] Position gelöscht, es sei den die aktuelle Position ist i == 0. Die letzte Spalte des Array wird benötigt um den String abzuschließen (0x00). Ohne diese Operation könnten auf dem Speicherinhalt keine komfortablen String-Funktionen angewandt werden.

4.3 Struktogram und Programmcode

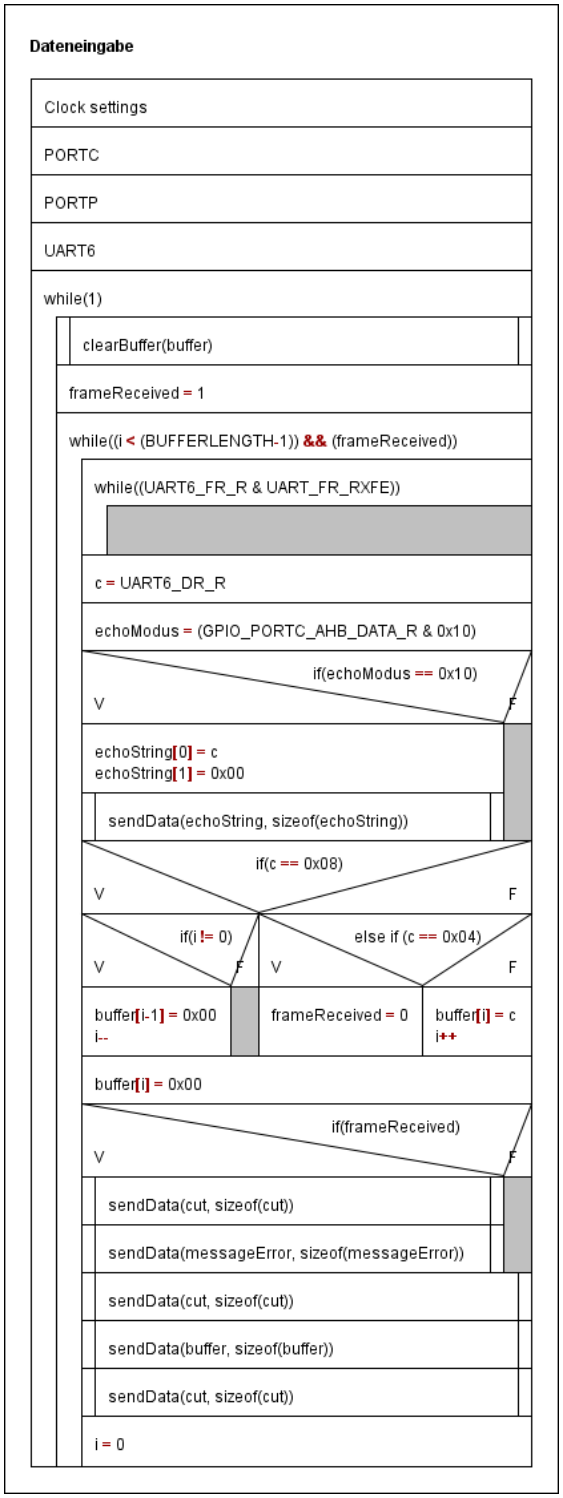

Abbildung 7: Nassi-Schneiderman Struktogram - Empfangsprogramm

```c
/*********************************************************************
      Autor: NPA
      Datum: 13.12.2016
      Dateiname: Dateeingabe
      Version: 1.0
 *********************************************************************/

#include <stdio.h>
#include <stdint.h>
#include "tm4c1294ncpdt.h"

#define BITRATE 9600
#define SYSCLK 25000000.0
#define BRD SYSCLK / (16 * BITRATE)
#define IBRD (int) BRD
#define FBRD (int) (((BRD-IBRD) * 64) + 0.5 )

#define BITDATA 0x54
#define WAITLOOPLENGTH 20000

// length data buffer for input data
#define BUFFERLENGTH 16

void clearBuffer(char buffer[BUFFERLENGTH]);
void sendData(char data[], int length);

void main(void) {

      uint8_t wt=0;
      uint32_t i=0;

      /*****************************
       * CLOCK - Settings
       *****************************/
      // switch over to main quartz oscillator at 25MHz
      // clear MOSC power down, high oscillator range setting,
      // and no crystal present setting
      SYSCTL_MOSCCTL_R &= ~(SYSCTL_MOSCCTL_OSCRNG | SYSCTL_MOSCCTL_PWRDN |
      SYSCTL_MOSCCTL_NOXTAL);
      // increase the drive strength for MOSC
      SYSCTL_MOSCCTL_R |= SYSCTL_MOSCCTL_OSCRNG;
      // set the main oscillator as main clock
      SYSCTL_RSCLKCFG_R = SYSCTL_RSCLKCFG_OSCSRC_MOSC;

      /*****************************
       * PORTC - Define
       *****************************/
      SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R2; wt++;
      GPIO_PORTC_AHB_DEN_R = 0x10;
      GPIO_PORTC_AHB_DIR_R = 0x00;

      /*****************************
       * PORTP - Define
       *****************************/
      SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R13; wt++;
      GPIO_PORTP_DEN_R |= 0x03;
      GPIO_PORTP_DIR_R |= 0x02;
      // alternative pin function PP1 -> TxD6, PP0 -> RxD6
      GPIO_PORTP_AFSEL_R |= 0x03;
      // controls the MUX for a pin example PP.1 -> TxD6
      // each nipple (4-Bit) configure Output Bit
      GPIO_PORTP_PCTL_R |= 0x00000011;

```

```
64      /*****************************
65       * UART6 - Define
66       *****************************/
67      // enable clock at UART module 6
68      SYSCTL_RCGCUART_R |= SYSCTL_RCGCUART_R6; wt++;
69      // disable UART during initialization
70      UART6_CTL_R &= ~UART_CTL_UARTEN;
71      // configure BAUD rate
72      // set DIVINT of BRD
73      UART6_IBRD_R = IBRD;
74      // set DIVFRAC of BRD
75      UART6_FBRD_R = FBRD;
76      // set Frame-Parameter
77      // WLEN_x = bits
78      // PEN 0 parity disable 1 enable
79      // EPS 0 odd parity 1 even
80      // STP2 0 one stop bit 1 two stop bits
81      UART6_LCRH_R |= (UART_LCRH_WLEN_7 | UART_LCRH_PEN | UART_LCRH_EPS);
82      // after initialization re-enable UART0
83      UART6_CTL_R |= (UART_CTL_UARTEN | UART_CTL_TXE | UART_CTL_RXE); wt++;
84
85      char c, buffer[BUFFERLENGTH];
86      char messageError[] = "Data overflow!";
87      char cut[] = "\n\r";
88      char echoString[2];
89      uint8_t frameReceived=0;
90      uint8_t echoModus=0;
91
92      i=0;
93      while(1){
94            clearBuffer(buffer);
95            frameReceived = 1;
96            // loop while buffer not full
97            while((i < (BUFFERLENGTH-1)) && (frameReceived)){
98                  // wait for flag: WHILE receive FIFO empty
99                  while((UART6_FR_R & UART_FR_RXFE));
100                 // = till a data frame are received
101                 c = UART6_DR_R; // read byte from UART6 data register
102
103                 echoModus = (GPIO_PORTC_AHB_DATA_R & 0x10);
104
105                 if(echoModus == 0x10){
106                       echoString[0] = c;
107                       echoString[1] = 0x00;
108                       sendData(echoString, sizeof(echoString));
109                 }
110
111                 // console input for BACKSPACE
112                 // 0x08 -> backspace
113                 if(c == 0x08){
114                       printf("BACKSPACE\n");
115                       if(i!=0) {
116                             buffer[i-1] = 0x00;
117                             i--;
118                       }
119                       // console input for EOT = Strg+D
120                 } else if(c == 0x04) {
121                       frameReceived = 0;
122                 } else {
123                       buffer[i]=c;
124                       i++;
125                 }
126           }
127           // Set 0 (Zero character) at the and of the string
```

```
128             buffer[i] = 0x00;
129
130             if(frameReceived) {
131                   sendData(cut, sizeof(cut));
132                   sendData(messageError, sizeof(messageError));
133             }
134
135             sendData(cut, sizeof(cut));
136             sendData(buffer, sizeof(buffer));
137             sendData(cut, sizeof(cut));
138
139             printf("\n Content of Data Buffer \n%s\n",buffer);
140             i=0;
141       }
142 }
143
144 void clearBuffer(char buffer[BUFFERLENGTH]) {
145       uint32_t i=0;
146       for(i=0; i<BUFFERLENGTH; i++) {
147             buffer[i] = 0x00;
148       }
149 }
150
151 void sendData(char data[], int length) {
152       uint32_t i=0;
153       for(i=0; i<length; i++) {
154             UART6_DR_R = data[i];
155             while(!(UART6_FR_R & UART_FR_TXFE));
156       }
157 }
```

Listing 3: Programmcode Empfangsprogramm

4.4 Fazit

Bei der verwendeten Bit-Rate ist eine Verzögerungszeit zwischen der Eingabe und der Ausgabe von mehreren ASCII-Zeichen bemerkbar, sodass sich diese Konfiguration nicht für eine schnelle Datenübermittlungen eignet.

Literatur

[1] K. R. Riemschneider HAW Hamburg. *07 Foliensatz Serielle Kommunikation UART Handout http://www.elearning.haw-hamburg.de/mod/resource/view.php?id=673295.* 2016.

[2] N.N. *Tiva^{TM} TM4C1294NCPDT Microcontroller DATA SHEET.* Texas Instruments Incorporated, ds-tm4c1294ncpdt-15863.2743 spms433b edition, 2014.