



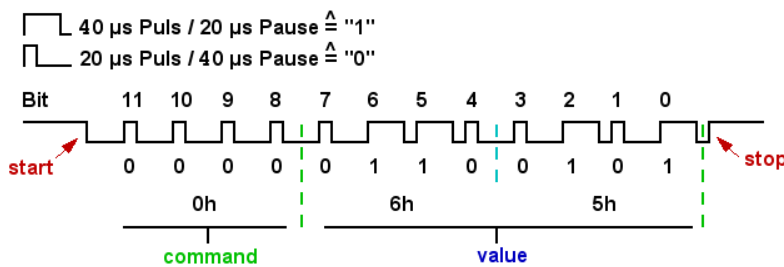
## Datenframe der proprietären PWM-Version

Im Ruhezustand (es wird nichts gesendet) muß die Datenleitung auf high gehalten werden. Ein Wechsel von high nach low stellt das Startbit dar. Nachdem das Startbit gesendet wurde muß innerhalb von 500µs mit dem Senden von insgesamt 12 Datenbits begonnen werden.

Ein einzelnes Datenbit besteht aus einer PWM. Eine Pulsdauer von weniger als 40µs repräsentiert eine logische 0, eine Pulsdauer von mehr als 40µs, höchstens 500µs repräsentiert eine logische 1.

Ein Datenframe besteht aus insgesamt 12 Bits, 4 Bits für ein Displaykommando und 8 Datenbits für einen Datenwert.

### 12-Bit Datenframe

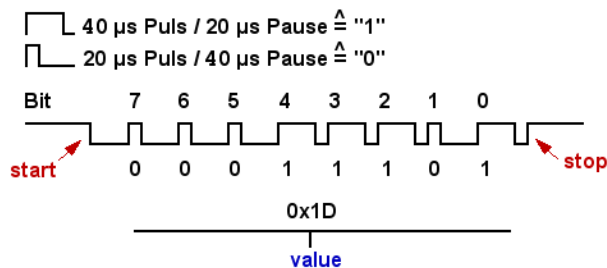


Command 0h mit Wert 65h schreibt ein kleines "a" (Ascii-Code 65h) auf das Display an der aktuellen Ausgabeposition

Nebenstehendes Impulsdiagramm zeigt ein Datenframe, das auf dem Display den Buchstaben „a“ anzeigt.

Die höherwertigen 4 Bits beinhalten das Displaykommando, in diesem Fall das Kommando 0 (für Ausgabe Ascii-Zeichen) und den unteren 8 Bits, die den Dateninhalt, das Ascii-Zeichen selbst darstellen (hier der Buchstabe "a").

### 8-Bit Datenframe



Einen Sonderfall stellt das Displaykommando 4 (set user char) dar. Diesem Kommando müssen weitere 8 Datenwerte folgen, die das Bitmap des zu setzenden Zeichens enthalten. Hier erwartet der Adapter dann ein Datenframe von nur 8 anstelle 12 Bits (siehe nebenstehendes Impulsdiagramm).

## Steuerkommandos PWM-Version

Bit	11	10	9	8	7	6	5	4	3	2	1	0				
	command-nibble				value											
	Bedeutung															
	0x00				<p>set char</p> <p>gibt ein Ascii-Zeichen mit dem Wert in value an der aktuellen Position aus.</p>											
	0x01				<p>gotoxy</p> <p>legt die Position für das nächste auszugebende Zeichen fest.</p> <table border="1"> <tr> <td>Bit 7:5</td> <td>y-Position</td> </tr> <tr> <td>Bit 4:0</td> <td>x-Position</td> </tr> </table> <p>Position 1,1 ist links oben</p>								Bit 7:5	y-Position	Bit 4:0	x-Position
Bit 7:5	y-Position															
Bit 4:0	x-Position															
	0x02				<p>clrscr</p> <p>löscht den Inhalt des Displays durch erneutes Initialisieren.</p>											
	0x03				<p>set additional char</p> <p>Im Programmcode des Displayadapters sind in einem Array definierte Bitmaps für Benutzerzeichen hinterlegt, die im Display dem Zeichensatz hinzugefügt werden können. Diese Zeichen können unter Ascii-Code 0..7 auf das Display gebracht werden können.</p> <table border="1"> <tr> <td>Bit 7:3</td> <td>Nummer (Element) des vordefinierten Zeichens im Array (siehe vordefinierte Zeichen)</td> </tr> <tr> <td>Bit 2:0</td> <td>Ascii-Code (0x00 .. 0x07) unter der das Zeichen abgerufen werden kann</td> </tr> </table>								Bit 7:3	Nummer (Element) des vordefinierten Zeichens im Array (siehe vordefinierte Zeichen)	Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen abgerufen werden kann
Bit 7:3	Nummer (Element) des vordefinierten Zeichens im Array (siehe vordefinierte Zeichen)															
Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen abgerufen werden kann															
	0x04				<p>set user char</p> <p>erwartet im Anschluss an dieses Kommando weitere 8 Datenbytes, die als benutzerdefiniertes Bitmap dem Zeichensatz des Displays hinzugefügt wird.</p> <table border="1"> <tr> <td>Bit 2:0</td> <td>Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann</td> </tr> </table> <p>Die nachfolgend zu sendenden Zeichen haben dasselbe Timing wie das Datenframe, es dürfen per Frame jedoch nur 8 Bits gesendet werden.</p>								Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann		
Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann															

<p style="text-align: center;">0x05</p>	<p>display shift left</p> <p>verschiebt das Display um eine Anzahl Zeichen nach links</p> <table border="1" data-bbox="638 324 1361 398"> <tr> <td style="width: 100px;">Bit 7:0</td> <td>Anzahl der Zeichen, um die der Textinhalt nach links verschoben wird</td> </tr> </table>	Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach links verschoben wird
Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach links verschoben wird		
<p style="text-align: center;">0x06</p>	<p>display shift right</p> <p>verschiebt das Display um eine Anzahl Zeichen nach rechts</p> <table border="1" data-bbox="638 633 1361 707"> <tr> <td style="width: 100px;">Bit 7:0</td> <td>Anzahl der Zeichen, um die der Textinhalt nach rechts verschoben wird</td> </tr> </table>	Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach rechts verschoben wird
Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach rechts verschoben wird		
<p style="text-align: center;">0x07</p>	<p>set contrast</p> <p>erlaubte Werte sind 0x00 .. 0xFF. 0x00 repräsentiert 0V (max. Kontrast), 0xFF repräsentiert Vcc (min. Kontrast, Text meistens nicht mehr lesbar)</p>		

## Steuerkommandos UART-Version

Byte 0: Kommandobyte	Byte 1: Datenbyte				
<p>0x00</p> <p>set char</p>	<p>auszugebendes Ascii-Zeichen</p> <p>Erlaubte Werte sind 0..127, da Bit 7 in der UART-Version die Kennung eines Kommandos ist. Somit sind die Zeichen von 128..255 nicht erreichbar, die jedoch bei den meisten Displays chinesische oder japanische Zeichen beinhalte</p>				
<p>0x01</p> <p>gotoxy</p>	<p>gotoxy</p> <p>legt die Position für das nächste auszugebende Zeichen fest.</p> <table border="1"> <tr> <td>Bit 7:5</td> <td>y-Position</td> </tr> <tr> <td>Bit 4:0</td> <td>x-Position</td> </tr> </table> <p>Position 1,1 ist links oben</p>	Bit 7:5	y-Position	Bit 4:0	x-Position
Bit 7:5	y-Position				
Bit 4:0	x-Position				
<p>0x02</p> <p>clrscr (löscht den Displayinhalt durch erneutes Initialisieren)</p>	<p>dem Kommandobyte 0x02 folgt kein weiteres Byte</p>				
<p>0x03</p> <p>set additional char</p>	<p>set additional char</p> <p>Im Programmcode des Displayadapters sind in einem Array definierte Bitmaps für Benutzerzeichen hinterlegt, die im Display dem Zeichensatz hinzugefügt werden können. Gleichzeitig können 8 verschiedene Zeichen hinzugefügt werden, die unter Ascii-Code 0..7 auf das Display gebracht werden können.</p> <table border="1"> <tr> <td>Bit 7:3</td> <td>Nummer (Element) des vordefinierten Zeichens im Array (siehe vordefinierte Zeichen)</td> </tr> <tr> <td>Bit 2:0</td> <td>Ascii-Code (0x00 .. 0x07) unter der das Zeichen abgerufen werden kann</td> </tr> </table>	Bit 7:3	Nummer (Element) des vordefinierten Zeichens im Array (siehe vordefinierte Zeichen)	Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen abgerufen werden kann
Bit 7:3	Nummer (Element) des vordefinierten Zeichens im Array (siehe vordefinierte Zeichen)				
Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen abgerufen werden kann				
<p>0x04</p> <p>set user char</p>	<p>erwartet im Anschluss an dieses Kommando weitere 8 Datenbytes, die als benutzerdefiniertes Bitmap dem Zeichensatz des Displays hinzugefügt wird.</p> <table border="1"> <tr> <td>Bit 2:0</td> <td>Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann</td> </tr> </table>	Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann		
Bit 2:0	Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann				

<p>0x05</p> <p>display shift left</p>	<p>verschiebt das Display um eine Anzahl Zeichen nach links</p> <table border="1"> <tr> <td>Bit 7:0</td> <td>Anzahl der Zeichen, um die der Textinhalt nach links verschoben wird</td> </tr> </table>	Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach links verschoben wird
Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach links verschoben wird		
<p>0x06</p> <p>display shift right</p>	<p>verschiebt das Display um eine Anzahl Zeichen nach rechts</p> <table border="1"> <tr> <td>Bit 7:0</td> <td>Anzahl der Zeichen, um die der Textinhalt nach rechts verschoben wird</td> </tr> </table>	Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach rechts verschoben wird
Bit 7:0	Anzahl der Zeichen, um die der Textinhalt nach rechts verschoben wird		
<p>0x07</p> <p>set contrast</p>	<p>erlaubte Werte sind 0x00 .. 0xFF. 0x00 repräsentiert 0V (max. Kontrast), 0xFF repräsentiert Vcc (min. Kontrast, Text meistens nicht mehr lesbar)</p>		

## Steuerzeichen

Wie bei einem Terminal interpretiert der Displayadapter besondere Ascii-Zeichen als Steuerzeichen für das Displays

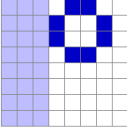
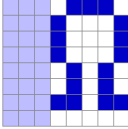
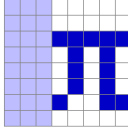
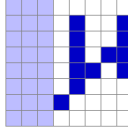
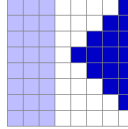
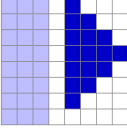
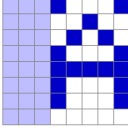
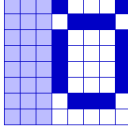
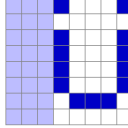
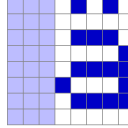
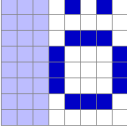
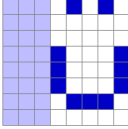
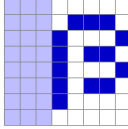
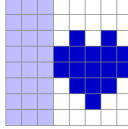
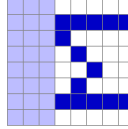
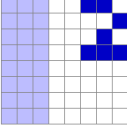
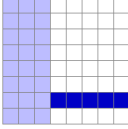
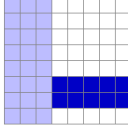
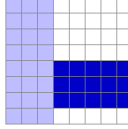
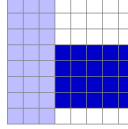
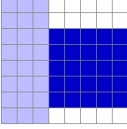
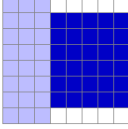
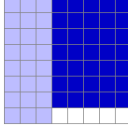
Dez	Hex	C	ISO	Bedeutung
8	0x08	\b	BS	backspace: setzt die Ausgabeposition für das nächste Zeichen um eine Stelle nach links
9	0x09	\t	HT	horizontal tab: setzt die Ausgabeposition für das nächste Zeichen auf die nächste vordefinierte Stelle in der aktuellen Zeile (Tabulatorstop, hier alle 4 Zeichen)
10	0x0A	\n	LF	line feed: setzt die Ausgabeposition für das nächste Zeichen auf die Zeile unter der aktuellen Zeile. Ist die aktuelle Zeile die unterste Zeile des Displays, wird diese Zeile um eine Position nach oben gescrollt. Die X-Ausgabeposition bleibt erhalten
13	0x0D	\r	CR	carriage return: setzt die Ausgabeposition für das nächste Zeichen auf den Zeilenanfang der aktuellen Zeile
14	0x0E		SO	hier: blinkenden Cursor an
15	0x0F		SI	hier: blinkenden Cursor aus

Hinweis: die Verwendung des Textdisplayadapters als Terminalanzeige ist nur bedingt, mit eigener Ausgabe jedoch sehr gut, verwendbar, da zwischen dem Senden einzelner Zeichen grundsätzlich eine 3 Millisekunden Zeitabstand eingehalten werden muß (diese Zeit wird benötigt, um das Zeichen zu verarbeiten und darzustellen).

Für das Verarbeiten eines Zeilenvorschubs "line feed" darf das nächste Zeichen erst nach 12 Millisekunden erfolgen. Diese Zeit wird benötigt, um ein evtl. notwendiges scrollen nach oben durchführen zu können.

# Vordefinierte Zeichen im Array < additional\_chars >

Bitmaps und deren "Aussehen", die im Array < additional\_chars > definiert sind

<p><b>Zeichen 0</b></p>  <pre> 0x0C 0x12 0x12 0x0c 0x00 0x00 0x00 0x00 </pre>	<p><b>Zeichen 1</b></p>  <pre> 0x0E 0x11 0x11 0x11 0x0A 0x0A 0x1B 0x00 </pre>	<p><b>Zeichen 2</b></p>  <pre> 0x00 0x00 0x1F 0x0A 0x0A 0x0A 0x13 0x00 </pre>	<p><b>Zeichen 3</b></p>  <pre> 0x00 0x09 0x09 0x0B 0x0D 0x08 0x10 0x00 </pre>	<p><b>Zeichen 4</b></p>  <pre> 0x01 0x03 0x07 0x0F 0x07 0x03 0x01 0x00 </pre>
<p><b>Zeichen 5</b></p>  <pre> 0x08 0x0C 0x0E 0x0F 0x0E 0x0C 0x08 0x00 </pre>	<p><b>Zeichen 6</b></p>  <pre> 0x11 0x04 0x0A 0x11 0x1F 0x11 0x11 0x00 </pre>	<p><b>Zeichen 7</b></p>  <pre> 0x11 0x0E 0x11 0x11 0x11 0x11 0x0E 0x00 </pre>	<p><b>Zeichen 8</b></p>  <pre> 0x11 0x00 0x11 0x11 0x11 0x11 0x0E 0x00 </pre>	<p><b>Zeichen 9</b></p>  <pre> 0x0A 0x00 0x0E 0x01 0x0F 0x11 0x0F 0x00 </pre>
<p><b>Zeichen 10</b></p>  <pre> 0x0A 0x00 0x0E 0x11 0x11 0x11 0x0E 0x00 </pre>	<p><b>Zeichen 11</b></p>  <pre> 0x0A 0x00 0x00 0x11 0x11 0x11 0x0E 0x00 </pre>	<p><b>Zeichen 12</b></p>  <pre> 0x00 0x0E 0x11 0x16 0x11 0x16 0x10 0x00 </pre>	<p><b>Zeichen 13</b></p>  <pre> 0x00 0x00 0x0A 0x1F 0x1F 0x0E 0x04 0x00 </pre>	<p><b>Zeichen 14</b></p>  <pre> 0x00 0x1F 0x10 0x08 0x04 0x08 0x1F 0x00 </pre>
<p><b>Zeichen 15</b></p>  <pre> 0x06 0x01 0x02 0x07 0x00 0x00 0x00 0x00 </pre>	<p><b>Zeichen 16</b></p>  <pre> 0x00 0x00 0x00 0x00 0x00 0x00 0x1F 0x00 </pre>	<p><b>Zeichen 17</b></p>  <pre> 0x00 0x00 0x00 0x00 0x00 0x00 0x1F 0x1F 0x00 </pre>	<p><b>Zeichen 18</b></p>  <pre> 0x00 0x00 0x00 0x00 0x00 0x00 0x1F 0x1F 0x00 </pre>	<p><b>Zeichen 19</b></p>  <pre> 0x00 0x00 0x00 0x00 0x1F 0x1F 0x1F 0x1F 0x00 </pre>
<p><b>Zeichen 20</b></p>  <pre> 0x00 0x00 0x1F 0x1F 0x1F 0x1F 0x1F 0x1F 0x00 </pre>	<p><b>Zeichen 21</b></p>  <pre> 0x00 0x1F 0x1F 0x1F 0x1F 0x1F 0x1F 0x1F 0x00 </pre>	<p><b>Zeichen 22</b></p>  <pre> 0x1F 0x1F 0x1F 0x1F 0x1F 0x1F 0x1F 0x1F 0x00 </pre>		

## Software des Textdisplayadapters

Sämtliche Software ist unter Linux entstanden und deshalb beziehen sich hier alle Angaben zum Compilieren und Linken auf Linux.

Die Firmware des Displayadapters findet sich im Archiv in folgenden Verzeichnissen

PWM-Version:                   ./stxlcd/pfs154/stxlcd\_recv  
UART-Version:                 ./stxlcd/pfs154/stxlcd\_recv\_uart

Innerhalb dieser Verzeichnisse kann die Firmware des Textdisplaysadapter mit einem einfachen

`make`

übersetzt und gelinkt werden. Voraussetzung hierfür ist ein Systemweit installierter Compiler SDCC ab Version 4.1 Es wird eine Binärdatei stxlcd\_recv.ihx / stxlcd\_recv\_uart.ihx erzeugt, die mittels

`make flash`

in den Controller geflasht werden kann. Im Makefile ist hier einzustellen, welcher Programmierer zu verwenden ist, zur Auswahl stehen der `< easypdkprogrammer >` oder der im Forum [www.mikrocontroller.net](http://www.mikrocontroller.net) veröffentlichte Arduino basierender Programmierer.

Links:

easypdkprogrammer:           <https://github.com/free-pdk/easy-pdk-programmer-hardware>

Arduino basierender Programmierer:   <https://www.mikrocontroller.net/topic/508930>

Sämtliche zur Übersetzung und zum Flashen notwendigen Dateien befinden sich in den entsprechenden Unterverzeichnissen (siehe auch Anhang "Verzeichnisstruktur der Dateien im Archiv").



## Transmittersoftware

Um den Textdisplayadapter ansteuern zu können, stehen Quelldateien für folgende Mikrocontrollerfamilien zur Verfügung:

Controllerfamilie	Verzeichnis im Archiv
STM8	./stxlcd/stm8_pwm ./stxlcd/stm8_uart
AVR	./stxlcd/avr_pwm ./stxlcd/avr_uart
Padauk PFS154	./stxlcd/pfs154/stx_lcd_transmit
STM32F0	./stxlcd/stm32f0

Auch hier gilt: Die Software ist unter Linux entstanden und ein einfacher Aufruf von `-- make --` im entsprechenden Archiv generiert ein flashbares Hex-Demoprogramm.

Hier wird vorausgesetzt, dass die entsprechenden Compiler systemweit installiert sind:

für PFS154 und STM8:      `sdcc 4.1` oder höher  
für AVR:                    `avr-gcc 7.3.0` oder höher  
für STM32F0:                `arm-none-eabi-gcc 6.2.1` oder höher

Steuerungspins der Controller:

- STM8:                    PA3, bei einem Controller im TSSOP-20 Gehäuse ist das der Pin 10
- AVR:                     PD2, bei einem ATmegax8 im 28 pol. Gehäuse ist das der Pin 2
- PFS154:                 PA4, bei einem 16 pol. Gehäuse Pin 10, bei einem 8 pol. Gehäuse Pin 6
- STM32F030:             PA9 (USART1\_TX), bei einem Controller im TSSOP-20 Gehäuse ist das der Pin 17

Für alle 4 Mikrocontrollerfamilien sind in den entsprechenden Verzeichnissen in den Unterordnern `<include>` und `<src>` die Quellcodes zur Steuerung des Displayadapters enthalten:

- `stxlcd.h` oder `stxlcd_uart.h` in `< include>`
- `stxlcd.c` oder `stxlcd_uart.c` in `< src >`

## Funktionen, Variable und Makros von stxlcd.c und stxlcd\_uart.c

Für alle genannten Controller existieren folgende C-Funktionen im Quellcode. Diese Funktionen sind für beide Versionen, PWM oder UART, gleichermaßen verfügbar.

### **stxt\_init();**

initialisiert die Ausgabepins des Controllers und, je nach Version ggf. einen UART-Port, der für das Senden zuständig ist.

### **stxt\_contrast(uint8\_t val);**

setzt die Kontrastspannung für das Textdisplay. Der Textdisplayadapter wird diesen Wert in eine PWM umsetzen.

Übergabe:

val: Kontrastwert des Displays. Erlaubte Werte sind im Bereich von 0..255. Hier entspricht 0 dem maximalen Kontrast und 255 minimalem Kontrast (die meistens Displays sind bei diesem Wert nicht mehr ablesbar. Für mehrere Displays haben sich gute Werte von 15 bis 40 herauskristalisiert.

### **stxt\_clrscr();**

sendet das Kommando zum Neuinitialisieren des Displays und löscht somit den Textinhalt

### **stxt\_sendval(uint16\_t val);** (nur PWM-Version)

sendet einen 12-Bit Wert an den Displayadapter. Protokoll siehe auch "Datenframe der proprietären PWM-Version"

Übergabe:

val: zu sendender 12-Bit Wert

### **stxt\_putchar(uint8\_t ch);**

sendet ein Ascii-Zeichen und gibt dieses an der aktuellen Ausgabeposition auf dem Display aus.

Übergabe:

ch: auszugebendes Zeichen (ch = Ascii-Code des Zeichens)

### **gotoxy(uint8\_t x, uint8\_t y);**

Positioniert die Ausgabeposition (x-y Koordinate), an der das nächste Zeichen ausgegeben wird. Die Koordinate 1,1 bezeichnet die linke obere Ecke.

Übergabe:

x, y : Koordinate, auf die die Ausgabeposition gesetzt wird.

### **stxt\_senduserchar(uint8\_t nr, const uint8\_t \*userch);**

sendet das Bitmap eines Benutzerzeichens, das dem Zeichensatz des Displays hinzugefügt wird.

Übergabe:

nr :               Ascii-Code Nummer unter der das Zeichen aufgerufen werden kann.  
\*userch:           Zeiger auf ein Array, das das Bitmap des Zeichens enthält

Beispiel:

```
// Bitmap des Eurozeichens
const uint8_t userchar[8] = { 0x07, 0x08, 0x1e, 0x08, 0x1e, 0x08, 0x07, 0x00 };

stxt_senduserchar(1, &userchar[0]);
gotoxy(1,2); stxt_putchar(1);
```

### **stxt\_addch(uint8\_t ch, uint8\_t pos);**

setzt ein Bitmap aus dem Array < additional\_chars > im Zeichensatz des Displays ein

Übergabe:

ch:     Zeichennummer (Element) des Arrays (siehe Vordefinierte Zeichen im  
        Array < additional\_chars >  
pos:     Ascii-Code Nummer unter der das Zeichen aufgerufen werden kann

Beispiel:

```
// Zeichen 15 im Array ist Quadratzeichen im Array und wird als
// Ascii-Code 1 dem Zeichensatz hinzugefügt

uint8_t z= 9;
stxt_addch(15, 1);
stxt_printf("%d%c= %d",z, 1, z*z);

// Auf dem Display wird angezeigt: 92= 81
```

### **stxt\_shiftleft(uint8\_t anz);**

verschiebt den Inhalt des Textdisplays um Anzahl < anz > Stellen nach links

Übergabe:

anz:     Anzahl der Stellen, um die der Inhalt des Displays nach links verschoben wird

### stxt\_shiftright(uint8\_t anz);

verschiebt den Inhalt des Textdisplays um Anzahl < anz > Stellen nach rechts

Übergabe:

anz: Anzahl der Stellen, um die der Inhalt des Displays nach rechts verschoben wird

### stxt\_printf(const uint8\_t \*s,...)

Um den Displayadapter einfach ansteuern zu können, wurden für die MCU-Familien AVR, STM8 und STM32F0 eine eigene, **printf** Funktion implementiert. Dieses **printf** ist zum einen abgespeckt worden (bspw. fehlt die Ausgabe von float-Werten), zum anderen jedoch für die Benutzung in Verbindung mit dem Adapter erweitert.

Grundsätzlich ist es mit dieser printf Funktion möglich, das Display komplett zu steuern, auch wenn es hierbei zu einem "kryptischen" Aussehen kommen kann.

Aufgrund des sehr geringen Flash-Speichers und des noch geringeren RAM eines PFS154 (2 kWord Flash / 128 Byte Ram) ist stx\_print für einen PFS154 **NICHT** definiert.

Folgende Platzhalter / Umwandlungszeichen sind in **stx\_printf** implementiert:

%s	Ausgabe eines Textstrings
%d	Ausgabe eines dezimalen Zahlenwertes. Die Ausgabe ist auf einen int16_t begrenzt
%x	Hexadezimale Ausgabe eines uint16_t Wertes. Ist der Wert größer 0xFF erfolgt die Ausgabe 4-stellig, ist der Wert kleiner-gleich 0xFF erfolgt die Ausgabe 2-stellig
%k	Ausgabe einer "Festkommazahl". Die globale Variable <b>printfkomma</b> bestimmt die Kommaposition der Ausgabe. Für printfkomma == 1 erfolgt für <pre>stx_printf("Wert: %k",876);</pre> die Anzeige: Wert: 87.6
%c	Ausgabe eines Ascii-Zeichens. Auch und vor allem anwendbar für benutzerdefinierte Zeichen
%p	setzt die X-Koordinate für die Ausgabe des nächsten Zeichens in der aktuellen Zeile
%P	setzt die Y-Koordinate für die Ausgabe des nächsten Zeichens in der aktuellen Spalte
%l	löscht den Inhalt der aktuellen Zeile und geht zu X-Position 1 für die Ausgabe des nächsten Zeichens

Grundsätzlich können die Steuerzeichen \n , \r, \t, \b in Verbindung mit **stx\_printf** verwendet werden.

Zum einfacheren Umgang mit stxt\_printf kann es sinnvoll sein, zu Beginn eines Programms diese Funktion auf ein printf im Programm mittels #define umzuleiten. Hier sollte dann im Programm die Bibliothek <stdio.h> **NICHT** eingebunden sein.

```
#define printf stxt_printf
```

Fortan kann dann mit printf Ausgaben auf das Display erfolgen

Beispiele:

```
printf("%P%l",2);           // gehe zu Zeile 2 und lösche diese
printf("%P%l",1);          // gehe zu Zeile 1 und lösche diese, Ausgabeposition
                             // ist jetzt 1,1 (links oben)
printf("Hallo\n\rWelt");   // in der ersten Zeile erscheint "Hallo", dann erfolgt
                             // ein linefeed \n gefolgt von einem carriage return
                             // und in der zweiten Zeile erscheint "Welt"
```

Natürlich können die Anweisungen auch innerhalb eines einzigen printf-Aufrufs erfolgen:

```
printf("%P%l%P%lHallo\n\rWelt",2,1);
```

Die Anweisungen

```
printf("%P%l%P%l%PHallo Welt",2,1,3);
```

positioniert die Ausgabe "Hallo Welt" ab der 3 Spalte in Zeile 1

Obige Ausgabe kann natürlich (lesbarer) auch erreicht werden mittels:

```
stxt_clrscr();             // Display löschen
gotoxy(3,1);
printf("Hallo Welt");
```

Verwendung von printf mit Tabulator und Festkommazahlen:

```
uint16_t w1, w2;

stxt_clrscr();
printfkomma= 2;
w1= 1634;
w2= 7631;
gotoxy(1,1);
printf("Wert 1\tWert 2");
printf("\n\r%k\t%k", w1, w2);
```

Anzeige auf dem Display:

```
Wert 1  Wert 2
16.34   76.31
```

## Ausgabefunktion für Padauk PFS154

Aufgrund des Speichermangels (insbesondere den nur 128 Byte großen RAM) wurde auf ein *printf* für den PFS154 verzichtet.

Stattdessen stehen für die Ausgabe die Funktion *stxt\_puts* und ein Integer to String-Konverter *stxt\_itoa* zur Verfügung.

```
puts(char *p);
```

gibt einen Ascii-Z String aus. Um Benutzerdefinierte Zeichen, das Anführungszeichen und das Prozentzeichen ausgeben zu können, nimmt das Prozentzeichen < % > ähnlich einem *printf* eine Ersetzung bei der Ausgabe durch.

%0 bis %7 wird durch ein Benutzerdefiniertes Zeichen mit dem Ascii-Code 0 .. 7 ersetzt.

%% gibt das Prozentzeichen aus

%a gibt Anführungsstriche aus

Übergabe:

\*p : Zeiger auf String

Beispiel:

```
puts("Hallo Welt");
```

```
char puffer[] = "PFS154";  
puts(&puffer[0]);
```

```
// dt. Umlaut 'a' (9. Position im vordefinierten Array)  
// an Speicherstelle 1 setzen
```

```
stxt_addch(9,1); // dt. Umlaut a an Speicherstelle 1  
puts("Z%1hler");
```

```
stxt_itoa(int i, char komma, char *puffer);
```

itoa = "*IntegerToAscii*"

konvertiert einen 16-Bit Integerwert zu einem AsciiZ-String. Da *stxt\_itoa* ebenfalls Festkommazahlen durch Einfügen eines Kommas an einer vom Benutzer wählbaren Position wandeln kann, kann ein erzeugter String max. 5 Ziffern, ein Kommazeichen sowie ein Minus-Vorzeichen enthalten. Zusammen mit einem abschließenden Zerobyte als Endekennung muß ein Pufferspeicher deshalb 8 Zeichen aufnehmen können.

Übergabe:

i: zu konvertierender Integerwert

komma: Position, an der ein Kommazeichen eingefügt werden soll

\*puffer: Zeiger auf einen Speicherbereich, der den String aufnehmen soll.

Beispiel:

```
char strpuffer[8];           // Puffer, der den konvertierten String
                             // aufnimmt
int16_t val;

stxt_clrscr();
val= -13278;

stxt_itoa(val,2,&strpuffer[0]); // 2 Kommastellen
stxt_puts("Wert: ");
stxt_puts(&strpuffer[0]);      // Ausgabe des Zahlenwerts
```

Auf dem Display wird: "Wert: -132.78" ausgegeben.

Für den, der den Textadapter brauchen kann: viel Spaß damit

R. Seelig im November 2024

## Anhang

### Verzeichnisstruktur der Dateien im Zip-Archiv

```
stxlcd
|
|-- /avr_pwm                               // Demo zum Ansteuern des Adapters PWM Version
|   |-- Makefile                           // Makefile zum Uebersetzen des Demoprogramms
|   |-- makefile.mk                         // wird von Makefile eingebunden
|   |-- avr_gpio.h                         // Deklarationen zum Ansteuern der Portpins
|   |-- stxlcd.c                           // Sourcecode zum Ansteuern des Displayadapters
|   |-- stxlcd.h                           // Header zu stxlcd.c
|   |-- stxlcd_demo.c                      // Demoprogramm
|   |-- stxlcd_demo.hex                    // flashbare Binaerdatei im Hex-Format
|
|-- /avr_uart                               // Demo zum Ansteuern des Adapters PWM Version
|   |-- Makefile                           // Makefile zum Uebersetzen des Demoprogramms
|   |-- makefile.mk                         // wird von Makefile eingebunden
|   |-- avr_gpio.h                         // Deklarationen zum Ansteuern der Portpins
|   |-- softuart.c                         // Sourcecode fuer softwarerealisiertes UART-Interface
|   |-- softuart.h                         // Header zu softuart.h
|   |-- stxlcd_uart.c                      // Sourcecode zum Ansteuern des Displayadapters
|   |-- stxlcd_uart.h                     // Header zu stxlcd_uart.c
|   |-- stxlcd_demo_uart.c                // Demoprogramm
|   |-- stxlcd_demo_uart.hex              // flashbare Binaerdatei im Hex-Format
|
|-- /doku
|   |-- txtlcd_adapter_v2.pdf              // PDF-Dokumentation
|
|-- /pfs154
|   |-- /stxlcd_recv                       // Firmware PWM-Version
|       |-- /include
|           |-- /easy-pdk                  // Verzeichnis, enthaelt Konfiguration fuer PFS und Programmier
|           |-- /pdk                       // Verzeichnis, enthaelt Deklarationen (u.a. Adressen der Peripherie
|                                           // lt. Datenblatt)
|           |-- delay.h                    // Header fuer Programmverzoegerungen
|           |-- pdk_init.h                 // Header zur Initialisierung des Systems
|           |-- pfs1xx_gpio.h              // Header zum Ansprechen der Portpins des PFS154
|       |-- /src
|           |-- delay.c                    // Sourcecode fuer Programmverzoegerungen
|       |-- /tools
|           |-- easypdkprog                 // Flashsoftware fuer EasyPdkProgrammer (Linux 64)
|           |-- pfsprog                     // Flashsoftware fuer Arduino basierenden Programm (Linux 64)
|           |-- pfsreadhex                  // liest aus einer *.ihx Datei den Speicherbedarf
|       |-- Makefile                       // Makefile zum Uebersetzen der Firmware PWM-Version
|       |-- makefile.mk                    // wird von Makefile eingebunden
|       |-- stxlcd_recv.c                  // Firmware PWM-Version
|       |-- stxlcd_recv.ihx                // flashbare Binaerdatei im Hex-Format
```



```

|         |
|         |     +.. userchars.txt           // Entwurf der Benutzerbitmaps als Ascii-Text-Datei
|         |
|         | +-- /stxlcd_recv_uart           // Firmware UART-Version
|         | |
|         | |     +-- /include
|         | | |
|         | | |     +-- /easy-pdk          // Verzeichnis, enthaelt Konfiguration fuer PFS und Programmier
|         | | |     +-- /pdk              // Verzeichnis, enthaelt Deklarationen (u.a. Adressen der Peripherie
|         | | |     // lt. Datenblatt)
|         | | |     +-- delay.h            // Header fuer Programmverzoegerungen
|         | | |     +-- pdk_init.h        // Header zur Initialisierung des Systems
|         | | |     +-- pfs1xx_gpio.h     // Header zum Ansprechen der Portpins des PFS154
|         | | |     +-- uart_2.h         // Header fuer softwarebasierendes UART-Interface
|         | |
|         | |     +-- /src
|         | | |
|         | | |     +-- delay.c           // Sourcecode fuer Programmverzoegerungen
|         | | |     +-- uart_2.c         // Sourcecode fuer softwarebasierendes UART-Interface
|         | |
|         | |     +-- /tools
|         | | |
|         | | |     +-- easypdkprog       // Flashsoftware fuer EasyPdkProgrammer (Linux 64)
|         | | |     +-- pfsprog          // Flashsoftware fuer Arduino basierenden Programm (Linux 64)
|         | | |     +-- pfsreadhex       // liest aus einer *.ihx Datei den Speicherbedarf
|         | |
|         | |     +-- Makefile            // Makefile zum Uebersetzen der Firmware PWM-Version
|         | |     +-- makefile.mk        // wird von Makefile eingebunden
|         | |     +-- stxlcd_recv_uart.c // Firmware PWM-Version
|         | |     +-- stxlcd_recv_uart.ihx // flashbare Binaerdatei im Hex-Format
|         | |     +.. userchars.txt     // Entwurf der Benutzerbitmaps als Ascii-Text-Datei
|         |
|         | +-- /stxlcd_transmit           // Demo zum Ansteuern des Adapters PWM Version
|         | |
|         | |     +-- /include
|         | | |
|         | | |     +-- /easy-pdk          // Verzeichnis, enthaelt Konfiguration fuer PFS und Programmier
|         | | |     +-- /pdk              // Verzeichnis, enthaelt Deklarationen (u.a. Adressen der Peripherie
|         | | |     // lt. Datenblatt)
|         | | |     +-- delay.h            // Header fuer Programmverzoegerungen
|         | | |     +-- pdk_init.h        // Header zur Initialisierung des Systems
|         | | |     +-- pfs1xx_gpio.h     // Header zum Ansprechen der Portpins des PFS154
|         | | |     +-- stxlcd.h          // Header fuer die Ansteuersoftware
|         | |
|         | |     +-- /src
|         | | |
|         | | |     +-- delay.c           // Sourcecode fuer Programmverzoegerungen
|         | | |     +-- stxlcd.c         // Sourcecode zum Steuern des Displayadapters
|         | |
|         | |     +-- /tools
|         | | |
|         | | |     +-- easypdkprog       // Flashsoftware fuer EasyPdkProgrammer (Linux 64)
|         | | |     +-- pfsprog          // Flashsoftware fuer Arduino basierenden Programm (Linux 64)
|         | | |     +-- pfsreadhex       // liest aus einer *.ihx Datei den Speicherbedarf
|         | |
|         | |     +-- Makefile            // Makefile zum Uebersetzen der Firmware PWM-Version
|         | |     +-- makefile.mk        // wird von Makefile eingebunden
|         | |     +-- stxlcd_demo.c      // Demoprogramm
|         | |     +-- stxlcd_demo.ihx   // flashbare Binaerdatei im Hex-Format
|         |
|         | + /stm8_pwm
|         | |
|         | |     +-- /include
|         | | |
|         | | |     +-- stm8_gpio.h      // Header zum Ansprechen der Portpins des STM8

```

```

|         |
|         | +-- stm8_init.h           // Header zur Systeminitialisierung
|         | |
|         | |-- stm8s.h              // Deklarationen zur Peripherie des STM8 (unvollstaendig)
|         | |
|         | +-- stxlcd.h             // Header fuer Ansteuersoftware stxlcd.c
|
| +-- /src
|         |
|         | +-- stm8_init.c
|         | |
|         | +-- stxlcd.c             // Sourcecode zum Ansteuern des Displayadapters
|
| +-- /tools
|         |
|         | +-- stm8flash            // Programersoftware zum Flashen von STM8 (Linux64)
|         | |
|         | +-- st8readihx          // liest aus einer *.ihx Datei den Speicherbedarf
|
| +-- Makefile                       // Makefile zum Uebersetzen des Demoprogramms
|
| +-- makefile.mk                    // wird vom Makefile eingebunden
|
| +-- stxlcd_demo.c                  // Demoprogramm
|
| +-- stxlcd_demo.ihx                // flashbare Binaerdatei im Hex-Format
|
+-- /stm8_uart
|
| +-- /include
|         |
|         | +-- stm8_gpio.h          // Header zum Ansprechen der Portpins des STM8
|         | |
|         | +-- stm8_init.h          // Header zur Systeminitialisierung
|         | |
|         | |-- stm8s.h              // Deklarationen zur Peripherie des STM8 (unvollstaendig)
|         | |
|         | +-- stxlcd_uart.h        // Header fuer Ansteuersoftware stxlcd_uart.c
|         | |
|         | +-- uart.h               // Header fuer UART-Interface
|
| +-- /src
|         |
|         | +-- stm8_init.c
|         | |
|         | +-- stxlcd_uart.c        // Sourcecode zum Ansteuern des Displayadapters
|         | |
|         | +-- uart.c               // Sourcecode fuer UART-Interface
|
| +-- /tools
|         |
|         | +-- stm8flash            // Programersoftware zum Flashen von STM8 (Linux64)
|         | |
|         | +-- st8readihx          // liest aus einer *.ihx Datei den Speicherbedarf
|
| +-- Makefile                       // Makefile zum Uebersetzen des Demoprogramms
|
| +-- makefile.mk                    // wird vom Makefile eingebunden
|
| +-- stxlcd_demo_uart.c            // Demoprogramm
|
| +-- stxlcd_demo_uart.ihx          // flashbare Binaerdatei im Hex-Format
|
+-- /stm32f0
|
| +-- /include
|         |
|         | +-- stxlcd_uart.h        // Header fuer Ansteuersoftware stxlcd_uart.c
|         | |
|         | +-- sysf0xx_init.h       // Header zum Initialisieren eines STM32F0 Systems
|         | |
|         | +-- uart.h               // Header fuer UART-Interface
|
| +-- /lib                            // Verzeichnis: libopencm3 Library
|
| +-- /src
|         |
|         | +-- stxlcd_uart.c        // Sourcecode zum Ansteuern des Displayadapters
|         | |
|         | +-- sysf0xx_init.c       // Sourcecode zur Initialisierung von STM32F0 Systemen
|         | |
|         | +.. uart.c               // Sourcecode fuer UART-Interface
|
| +-- /tools
|         |
|         | +-- stlink                // Verzeichnis, Quellcode fuer ST-Linkprogrammer (open source)
|         | |
|         | +-- dfu-util              // DFU-Programmer Software (Linux 64)

```

```
| |
| +.. stm32flash // Programmer fuer seriellen Bootloader (Linux 64)
|
|+-- Makefile // Makefile zum Uebersetzen des Quellprogramms benoetigt
| // makefile.mk im Verzeichnis ./lib
|
|+-- stxlcd_demo_uart.c // Demoprogramm
```