Arduino: Serielles Displayinterface für HD44780-Textdisplays

für ATmega-basierendes System

Prinzipiell besteht das „Serielle Displayinterface“ aus einem Firmwaresketch und einer Arduino-Library, die dieses Displayinterface (im Folgenden nur noch Interface benannt) ansteuern kann. Als Display können lediglich zweizeilige Displays verwendet werden.

Die Senderbibliothek übernimmt die Ausgabe von Zeichen und Steuerkommandos, während die Empfängerfirmware die empfangenen Daten interpretiert und das angeschlossene Display ansteuert.

Im Gegensatz zu den verbreiteten I²C-Interfaces auf Basis des PCF8574 befindet sich die komplette Ausgabelogik auf der Senderseite. Der Sender bestimmt nicht nur den auszugebenden Text, sondern auch Cursorpositionierung, Zeilenverwaltung, Sonderzeichen und weitere Displayfunktionen. Der Empfänger übernimmt lediglich die Interpretation der empfangenen Kommandos und die Ansteuerung des Displays.

Im Vergleich zu einer I²C-Lösung reduziert sich die Verbindung zwischen Sender und Interface auf eine einzige Datenleitung sowie eine gemeinsame Masseverbindung.

Die Anforderungen an dieses Interface waren bei der Entwicklung:

- Empfänger- und Sendesoftware sollen in Arduino realisiert werden
- Sender soll als Library verfügbar sein
- Übertragung soll mittels SoftwareSerial erfolgen

Für die Übertragung von Zeichen wurde – wie in den Anforderungen zu sehen – SoftwareSerial verwendet, damit der Hardware-UART eines ATmega für andere Aufgaben frei bleibt. Da das Interface prinzipiell nur zur Darstellung von Zeichen dient, aber selbst keine Zeichen senden kann, genügt empfängerseitig ein einzelner Receiverpin zur Datenkommunikation.

Das Protokoll des Interfaces lautet: **4800 Baud, 8 Datenbits, keine Parität, 1 Stopbit (8N1).**

Grundsätzlich könnte man das Interface auch als „Serielles Miniempfangsterminal“ bezeichnen, da es zum einen typische Steuerzeichen eines Strings interpretiert:

- Carriage return (\r)
- Line feed (\n)
- Tabulator (\t)
- Backspace (\b)

Und zusätzlich Steuerkommandos zum Positionieren der Cursorposition und Löschen einer einzelnen Zeile oder des ganzen Displays beherrscht.

Das Interface implementiert für zweizeilige Displays ein Scrollverhalten. Wird ein `\n` (Line Feed) empfangen, während sich der Cursor bereits in der zweiten Zeile befindet, wird der Displayinhalt um eine Zeile nach oben gescrollt. Die bisherige zweite Zeile wird dabei in die erste Zeile übernommen. Anschließend wird die zweite Zeile gelöscht und steht für weitere Ausgaben zur Verfügung.

Es wurde lange überlegt, ob aus der verwendeten Klasse des Empfängers eine Library gemacht werden soll, aber es wurde bewusst dagegen entschieden, da diese Klasse ausschließlich dem Zweck dient, dieses Interface bereitzustellen.

Die Firmwaredatei ist:

stxlcdrecv_uart.ino

Innerhalb dieser Datei ist eine Anpassung bezüglich der Anschlussbelegung Mikrocontroller und Display zu machen, damit ein fehlerfreier Betrieb möglich ist.

```
// Anschluesse Textdisplay an Arduino
static constexpr int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;

// Anschluss Receiver-Pin
static constexpr int rxPin = A4;

// ---------------------------------------------------------
// Objekt erzeugen
// ---------------------------------------------------------
StxUartReceive stxlcd(rs, en, d4, d5, d6, d7, rxPin);
```

Die Anschlusspins sind die Arduino-Namen und können (fast) frei gewählt werden. Die Klasse ***StxUartReceive*** erledigt alle Aufgaben, so dass „setup“ und „loop“ sehr klein ausfallen:

```
// ---------------------------------------------------------
//                          setup
// ---------------------------------------------------------
void setup()
{
  stxlcd.begin();
  stxlcd.clrscr();
  stxlcd.gotoxy(1,1);
  stxlcd.puts("STX-UART-LCD");
}

// ---------------------------------------------------------
//                          loop
// ---------------------------------------------------------
void loop()
{
  stxlcd.process();
}
```

Nach dem Flashen der Firmware erscheint in der ersten Zeile des Displays der Text

STX-UART-LCD

als Rückmeldung, dass das Interface betriebsbereit ist. Zur Beruhigung eines eventuell floatenden Eingangs kann an die Receive-Leitung ein PullUp-Widerstand von 22kΩ bis 47kΩ nach +Vcc geschaltet werden. Dieser ist bei fester Verdrahtung mit einem sendenden Mikrocontroller nicht notwendig.

Sollte, aus welchen Gründen auch immer, die Baudrate des Interfaces geändert werden, so geschieht dieses in der Methode:

```
// ---------------------------------------------------------
//                        begin
// ---------------------------------------------------------
void StxUartReceive::begin(void)
{
  LiquidCrystal::begin(16, 2);
  pinMode(_rxPin, INPUT);
  SoftwareSerial::begin(4800);
  linebufclr();
}
```

Versuche haben gezeigt, dass eine Baudrate von mehr als 9600 Bd nicht ratsam sind: hier können in seltenen Fällen Zeichenverluste auftreten. Bei geringerer Baudrate erscheint der Textaufbau etwas träge, so dass für diesen Zweck eine Baudrate von 4800 Bd als optimal angesehen wird.

Neben den vom Interface akzeptierten Steuerzeichen \n \r \t \b kann eine weitere Steuerung des Interfaces erfolgen.

Hierzu wertet das Interface Bit 7 des eingegangenen Zeichens aus. Ist dieses eine logische 0 wird das Zeichen als Ausgabezeichen interpretiert und an der aktuellen Cursorposition des Displays ausgegeben.

Ist Bit 7 eine logische 1 wird das empfangene Zeichen als Steuercode interpretiert und es wird ein zweites Byte erwartet, welches die Information des gewählten Kommandos trägt

Hinweis: Auf diese Weise sind alle Asciizeichen des Displays von 128 bis 255 nicht erreichbar, da hier Bit 7 gesetzt ist. Dieses sollte verschmerzbar sein, da die meisten HD44780-kompatiblen Displays im erweiterten Zeichenbereich ohnehin überwiegend japanische oder chinesische Schriftzeichen enthalten.

Kommandobytes

<table>
<tr><th>Byte 0: Kommandobyte</th><th>Byte 1: Datenbyte</th></tr>
<tr><td>0x81

gotoxy</td><td>gotoxy

legt die Position für das nächste auszugebende Zeichen fest.
<table><tr><td>Bit 7:5</td><td>y-Position</td></tr><tr><td>Bit 4:0</td><td>x-Position</td></tr></table></td></tr>
<tr><td>0x82

clrscr</td><td>dem Kommandobyte 0x82 folgt kein weiteres Byte</td></tr>
<tr><td>0x83

set additional char</td><td>set additional char

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.
<table><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></td></tr>
<tr><td>0x84

set user char</td><td>erwartet im Anschluss an dieses Kommando weitere 8 Datenbytes, die als benutzerdefiniertes Bitmap dem Zeichensatz des Displays hinzugefügt wird.
<table><tr><td>Bit 2:0</td><td>Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann</td></tr></table></td></tr>
<tr><td>0x85

display shift left</td><td>verschiebt das Display um eine Anzahl Zeichen nach links
<table><tr><td>Bit 7:0</td><td>Anzahl der Zeichen um der der Text nach links verschoben wird</td></tr></table></td></tr>
<tr><td>0x86

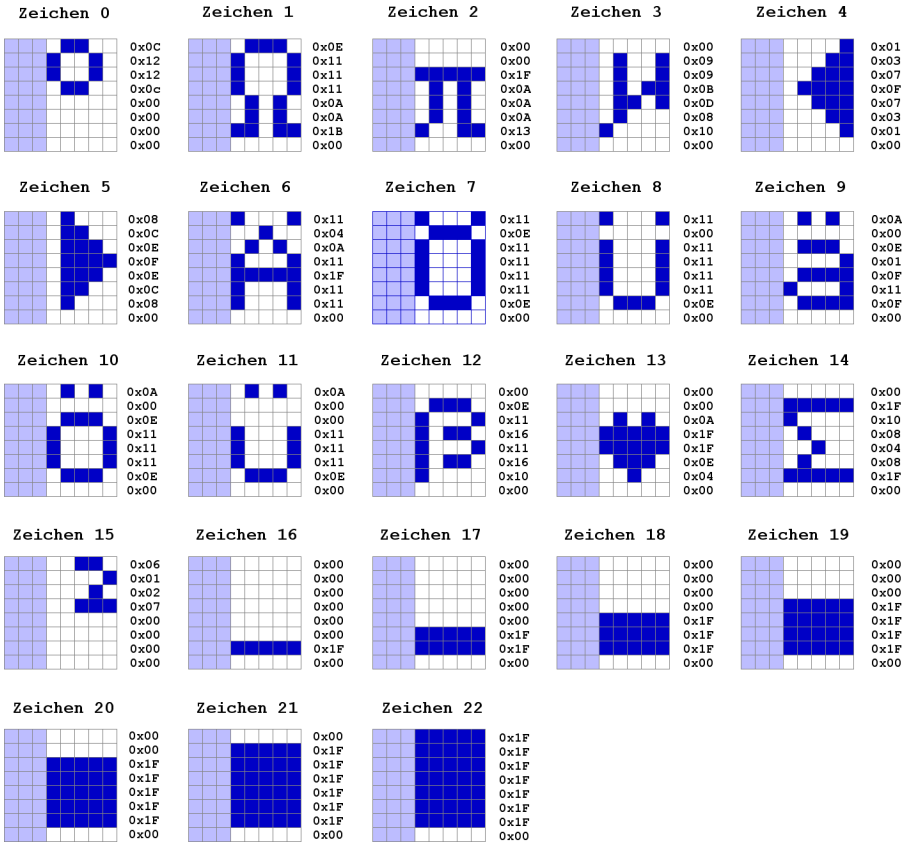
display shift right</td><td>verschiebt das Display um eine Anzahl Zeichen nach rechts
<table><tr><td>Bit 7:0</td><td>Anzahl der Zeichen um der der Text nach rechts verschoben wird</td></tr></table></td></tr>
</table>

Steuerzeichen

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

Vordefinierte Zeichen im Array <additional_chars>

Erscheinungsbild der definierten Bitmaps



Library
stxlcdsoftuart

Auch wenn das Display von jedem Device Bytes empfangen und anzeigen kann – beispielsweise von einem PC über eine USB2UART Brücke -, so empfiehlt es sich doch bei der Verwendung unter Arduino die Senderlibrary **stxlcdsoftuart** zu verwenden.

Mit dieser Library gestaltet sich eine Ausgabe auf dem LCD-Interface als ausgesprochen einfach. Es kann, wie von Arduino-Usern gewohnt, entweder über den Arduino-Stream oder / und mit einem auf das Interface abgestimmten und abgespeckten **printf-Version** betrieben werden.

Zudem ist u.a. die Positionierung des Cursors oder das Einsetzen von benutzerdefinierten Fontbitmaps hiermit möglich.

Die Library ist im Archiv als wiederum gepackte Datei vorhanden und kann innerhalb der Arduino-IDE mittels der Bibliotheksverwaltung installiert werden.

stxlcdsoftuart verwendet wie die Firmware des Interfaces selbst einen einzelnen Pin zur Datenübertragung an das Interface. Für die Benutzung muß eine Instanz der Klasse ***StxlcdUart*** instanziiert werden:

```
constexpr int txdPin = A4;   // Anschluss, auf dem Daten an das Displayinterface gesendet
                             // werden

// ------------------------------------------------------
//   Objekt erzeugen
// ------------------------------------------------------
StxlcdUart stxlcd(A4);
```

Beispiele zur Benutzung sind als Examples in der Library enthalten.

Klassenmethoden von StxlcdUart

Sämtliche Beispiele der folgenden Methodenbeschreibungen gehen davon aus, dass ein Objekt ***stxlcd*** (siehe oben) erzeugt worden.

begin(uint32_t baud)

Initialisiert serielle Übertragung mit der gewählten Baudrate baud und setzt den im Konstruktor angegeben Pin als Ausgang. Da das Empfänger-Displayinterface mit einer Baudrate von 4800 Bd betrieben wird, ist für baud hier 4800 anzugeben:

Beispiel:
```
stxlcd.begin(4800);
```

clrscr(void)

Clearscreen: Löscht das Display und setzt den Cursor an die Position 1,1 => links oben.

Beispiel:
```
stxlcd.clrscr();
```

lcdputchar(char ch)

Sendet ein Zeichen seriell und schreibt somit dieses Zeichen an die aktuelle Cursorposition im Display

Beispiel:
```
stxlcd.lcdputchar('A')    // schreibt das Zeichen 'A' auf das Display
```

gotoxy(uint8_t x, uint8_t y)

positioniert den Cursor auf dem Display. Position 1,1 repräsentiert die linke obere Ecke

Übergabe:

x,y : Koordinate an dieser der Cursor positioniert wird

Beispiel:

```
stxlcd.gotoxy(3,1);
stxlcd.print("Hallo");
stxlcd.gotoxy(3,2);Grundsätzlich kön
stxlcd.print("Welt");
```

senduserchar(uint8_t nr, const uint8_t *userch)

sendet das Bitmap eines Userzeichen bestehend aus 8 Bytes an das Displayinterface

Übergabe:

nr : Ascii-Codenummer, unter der das gesendete Bitmap als Zeichen abrufbar ist

*userch : Zeiger auf ein 8 Byte großes Array, das das Bitmap des Userzeichens enthält.

Beispiel (gibt das Eurozeichen an Position 1 in Zeile 2 aus):

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

stxlcd.senduserchar(1, &userchar[0]);
stxlcd.gotoxy(1,2);
stxlcd.putchar(1);
```

addch(uint8_t ch, uint8_t pos)

bindet ein im Displayinterface hinterlegtes Bitmap-Zeichen in den Zeichensatz des Displays ein.

Übergabe:

ch : Nummer des Bitmaps – siehe **„Vordefinierte Zeichen im Array <additional_chars>“** welches eingesetzt werden soll

pos : Ascii-Zeichen (0..7) unter der das Bitmap erreichbar ist

Beispiel (zeigt das Sigmazeichen an):

```
stxlcd.addch(14,3);
stxlcd.gotoxy(1,1);
stxlcd.print("Sigma-BMP: ");
stxlcd.print((char)(3));
```

shiftleft(uint8_t anz)

Displayinhalt nach links schieben

Übergabe:
anz : Anzahl der Zeichen um dass das Display nach links verschoben wird

shiftright(uint8_t anz)

Displayinhalt nach rechts schieben

Übergabe:
anz : Anzahl der Zeichen um dass das Display nach rechts verschoben wird

printf(const char *s, ...)

Um den Displayadapter einfach ansteuern zu können, wurde eigens hierfür eine eigene, sehr verschlankte ***printf*** Funktion implementiert. Die verschlankte printf-Version hier hat bspw. keine Ausgabe von float-Werten oder Feldlängen, wurde jedoch hinsichtlich des Steuerns des Interfaces um Funktionen erweitert

Folgende Platzhalter / Umwandlungszeichen sind in ***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 Klassenvariable printfkomma bestimmt die Kommaposition der Ausgabe. Grundsaetzlich wird hier ein Dezimalwert ausgegeben, aber an der durch printfkomma definierten Stelle ein Dezimalpunkt eingefügt. Stxlcd.printfkomma= 2; stxlcd.printf("Wert: %k",876); Anzeige auf dem Display: Wert: 8.76
%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 Zeichen

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

Beispiele:

```
  stxlcd.printf("%P%l",2);        // gehe zu Zeile 2 und lösche diese
  stxlcd.printf("%P%l",1);        // gehe zu Zeile 1 und lösche diese, Ausgabeposition
                                  // ist jetzt 1,1 (links oben)

  stxlcd.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:

```
  stxlcd.printf("%P%l%P%lHallo\n\r\Welt",2,1);
```

Die Anweisungen

```
  stxlcd.printf("%P%l%P%l%pHallo Welt",2,1,3);
```

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

Diese Ausgabe kann deutlich lesbarer auch erreicht werden durch:

```
  stxlcd.clrscr();                // Display loeschen
  stxlcd.gotoxy(3,1);
  stxlcd.printf("Hallo Welt");
```

Verwendung von printf mit Tabulator und Festkommazahlen:

```
  uint16_t w1, w2;

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

Resultierende Anzeige auf dem Display:

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