Intelligenter Textdisplayadatapter
mit Paudauk PFS154

Der Textdisplayadapter ist entstanden, um beim Experimentieren auf einem Steckbrett eine sehr einfache Möglichkeit zu haben, Text und Zahlen auszugeben.

Im Gegensatz zu bekannten Adaptern, die ein I²C Protokoll (welches 2 Kommunikationsleitungen, SDA und SCL benötigt) in parallele Daten wandelt, aber keinerlei Steuerung des Displays vornimmt, wird diesem Entwurf nur mittels einer einzigen Datenleitung übermittelt, was das Display anzeigen soll.

Eine sehr minimalistische (und äußerst preiswerte) Lösung ist das Ansteuern des Displays mittels einem Mikrocontroller PFS154 von Padauk.

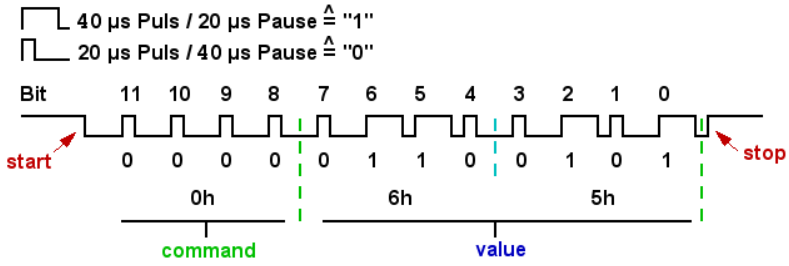
Ein proprietäres Datenformat, welches letztendlich eine Pulsweitenmodulation (PWM) darstellt, nimmt die Steuerbefehle entgegen und setzt diese in die Kommandos des Displays um.

Datenframe

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 hier 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, bestehend aus 4 Bits für ein Kommando an das Display und 8 Datenbits für einen Wert.



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 Kommando für das Display, in diesem Fall das Kommando 0 (welches das Kommando für das Ausgeben eines Ascii-Wertes ist) und den unteren 8 Bits, die den Dateninhalt (hier das Ascii-Zeichen) darstellen, der verarbeitet werden soll.

Steuerkommandos

Bit	11	10	9	8	7	6	5	4	3	2	1	0
	command-nibble				value							
					Bedeutung							
0x00					set char gibt ein Ascii-Zeichen mit dem Wert in value an der aktuellen Position aus.							
0x01					gotoxy legt die Position für das nächste auszugebende Zeichen fest. Bit 7:5 – y-Position Bit 4:0 – x-Position Position 1,1 ist links oben							
0x02					clrscr löscht den Inhalt des Displays durch erneutes Initialisieren.							
0x03					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. Bit 7:3 – Nummer (Element) des vordefinierten Zeichens im Array (siehe vordefinierte Zeichen) Bit 2:0 – Ascii-Code (0x00 .. 0x07) unter der das Zeichen dargestellt werden kann							

<table>
<tr><td>0x04</td><td>set user char

erwartet im Anschluss an dieses Kommando weitere 8 Datenbytes, die als benutzerdefiniertes Zeichen 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>

Die nachfolgend zu sendenden Zeichen haben dasselbe Timing wie das Datenframe, es dürfen per Frame jedoch nur 8 Bits gesendet werden.</td></tr>
<tr><td>0x05</td><td>display shift left

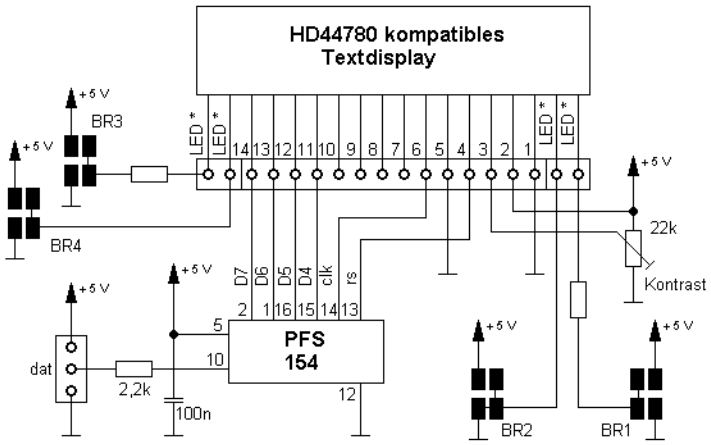
verschiebt das Display um eine Anzahl Zeichen nach links

<table><tr><td>Bit 7:0</td><td>Anzahl der Zeichen, um die der Textinhalt nach links verschoben wird</td></tr></table></td></tr>
<tr><td>0x06</td><td>display shift right

verschiebt das Display um eine Anzahl Zeichen nach rechts

<table><tr><td>Bit 7:0</td><td>Anzahl der Zeichen, um die der Textinhalt nach rechts verschoben wird</td></tr></table></td></tr>
</table>

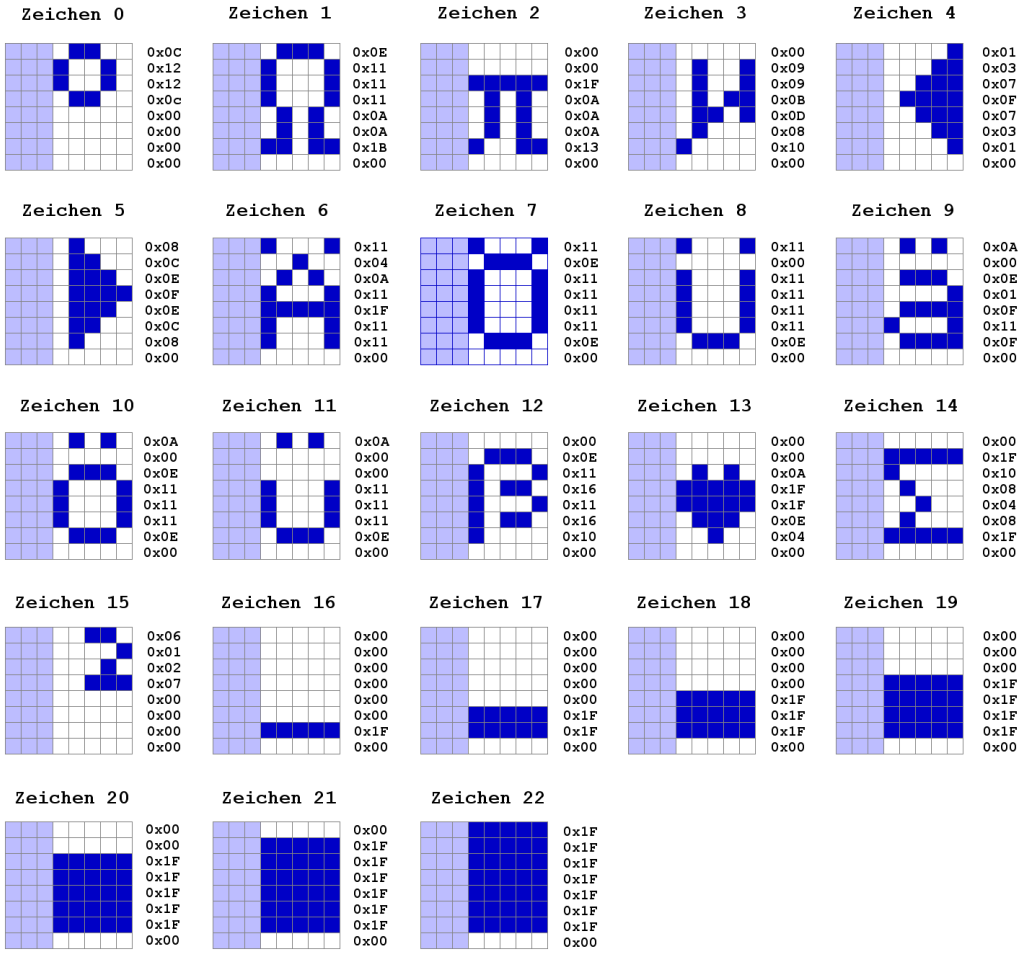
Schaltplan



* aufgrund unterschiedlicher Pinbelegungen der Textdisplays, bei denen die Pins der Beleuchtungsdioden rechts oder links der "regulären" Pins angeordnet sein können und auch hier keine einheitliche Zuordnung von Anode und Kathode gegeben ist, können diese Anschlüsse mittels der Lötbrücken BR1 bis BR4 konfiguriert werden.

Das Display wird mittels des Controllers im 4-Bit Modus betrieben. Die Dateneingangsleitung (Pin 10) ist der GPIO-Port PA4.

Vordefinierte Zeichen im Array < additional_chars >



Software des Textdisplayadapters

Die Software des Adapters ist im Archiv im Verzeichnis:

./stxlcd/pfs154/stxlcd_recv

zu finden.

Zum Übersetzen der Quelldatei wird ein im Linux-System installierter Compiler SDCC Version 4.1 oder höher benötigt. Ein einfaches

make

in diesem Verzeichnis erstellt die Binärdatei stxlcd_recv, die mittels

make flash

in den Controller geflasht werden kann. Im Makefile ist hier einzustellen, welcher Programmer zu verwenden ist, zur Auswahl stehen der < easypdkprogrammer > oder der im Forum www.mikrocontroller.net veröffentlichte Arduino basierender Programmer.

Links:

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

Arduino basierender Programmer: https://www.mikrocontroller.net/topic/508930

Transmittersoftware

Um den Textdisplayadapter ansteuern zu können, stehen Programme für die folgende Mikrocontroller-familien zur Verfügung:

- STM8 im Verzeichnis stxlcd/stm8 des Archivs
- AVR (ATmega und ATtiny) im Verzeichnis stxlcd/avr des Archivs
- Padauk PFS154 im Verzeichnis stxlcd/pfs154/stx_lcd_transmit des Archivs

Auch hier gilt: Die Software ist unter Linux entstanden und ein einfacher Aufrufe von -- make -- im entsprechenden Archiv generiert die dazugehörenden Binärdateien

Steuerungspins der Controller:

- STM8: PA3, bei einem Controller im TSSOP-20 Gehäuse ist das der Pin 10
- AVR: PD2, bei einem ATmegaxx8 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

Für alle 3 Mikrocontrollerfamilien ist in den entsprechenden Verzeichnissen in den Unterordnern < include > und < src> Software zur Steuerung des Displayadapters enthalten:

- stxlcd.h in < include>
- stxlcd.c in < src >

Für alle Controller gibt es dieselben Funktionen.

Funktionen, Variable und Makros von stxlcd

stxt_clrscr();

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

stxt_putchar(uint8_t ch);

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

gotoxy(uint8_t x, uint8_t y);

Positioniert die Ausgabeposition (x-y Koordinate), an der das nächste Zeichen ausgegeben wird. die Position

stxt_senduserchar(uint8_t nr, const uint8_t *userch);

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

nr : Ascii-Code Nummer unter der das Zeichen aufgerufen werden kann.
*userch: Zeiger auf ein Array, das die 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 < additional_chars > im Zeichensatz des Displays ein

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 2 im Array ist das pi - Zeichen und wird als
// Ascii-Code 1 dem Zeichensatz hinzugefügt
stxt_addch(2, 1);
gotoxy(1,2); stxt_putchar(1);
```

stxt_shiftleft(uint8_t anz);

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

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 links

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

zusätzliche Funktion beim PFS154

Die Demoprogramme für AVR und STM8 Mikrocontroller benutzen zur Ausgabe von Text ein sehr abgespecktes und minimalistisches printf-Derivat.

Selbst dieses ist auf dem PFS154 aufgrund des sehr geringen RAM des Controllers (nur 128 Byte) nicht praktikabel.

Aus diesem Grund wurde im Demoprogramm eine Funktion < puts > definiert, die hier beschrieben ist.

void 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 Anfuehrungsstriche aus

Uebergabe:

*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");
```

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

R. Seelig im Oktober 2024