

PicoMon

Terminalmonitor für CH32V003 und STM32F0xx Mikrocontroller

PicoMon ist eine Firmware für CH32V003 und STM32F0xx, die in Verbindung mit farbigen TFT-Displays und SPI-Interface einen Terminalmonitor darstellt.

PicoMon stellt die Zeichen auf dem Display dar, die über USART gesendet werden. Hierbei gilt folgendes Protokoll:

8 Datenbits, keine Parität, 1 Stopbit (8N1) mit 4800 Baud

Es können grafische Displays mit folgenden Auflösungen betrieben werden:

- 128 * 128 Pixel
- 160 * 80 Pixel
- 160 * 128 Pixel
- 320 * 240 Pixel

Hierbei werden folgende Displaycontroller unterstützt:

- ILI 9163
- ILI 9340
- ST 7735 / ST 7735R
- ST 7789
- S6D02A1
- ILI 9225

Für die Darstellung von Zeichen sind 4 verschiedene Schriftgrößen definiert:

- 5x7 Pixel (benötigt zur Darstellung auf dem Display 6x8 Pixel)
- 8x8 Pixel
- 10x16 Pixel
- 12x16 Pixel

PicoMon interpretiert einkommende Zeichen und ist in der Lage folgende Steuerzeichen zu interpretieren:

Dez	Hex	Ctrl	C	ISO	Funktion
8	0x08	^h	\b	BS	Backspace
9	0x09	^i	\t	HT	Horizontal-Tabulator
10	0x0A	^j	\n	LF	Line feed, Zeilensprung
13	0x0D	^m	\r	CR	Carriage return, gehe zum Zeilenanfang
16	0x10	^p		DLE	vom Standard abweichend: Werden nachfolgende Zeichen als Grafikanweisungen zum Zeichnen von Linen und Kreisen interpretiert
27	0x1B	^[ESC	leitet Escape-Sequenz ein

Erfolgt ein Linefeed in der letzten Zeile, erfolgt ein Scrollen des Textinhaltes um eine Zeile nach oben.

Escape-Sequenzen

PicoMon interpretiert eingehende Escape-Sequenzen hinsichtlich der Farben und der Cursorpositionierung als ANSI-Escape Sequenz.

Farbensteuerung

Ein `\033[` leitet die Escape-Sequenz ein. Dieser Einleitung können eine oder mehrere Zahlen folgen, die mit einem Semikolon `;` von einander getrennt sind. Ein abschließendes Zeichen `m` beendet die Sequenz.

Beispiel: Einstellen dunkelrote Schrift auf grauem Hintergrund
`printf("\033[0;31;47m");`

Hinweis: Helle Hintergrundfarben werden nicht unterstützt!

Farbcodes

Farbnummer	Textfarbe	Farbnummer	Hintergrundfarbe
0	dunkle Farbe		
1	helle Farbe		
30	schwarz	40	schwarz
31	rot	41	rot
32	grün	42	grün
33	braun / gelb	43	braun / gelb
34	blau	44	blau
35	lila	45	lila
36	türkis	46	türkis
37	grau	47	grau

Clear screen / lösche Display

Zum Löschen des Displayinhalts und zum Setzen des Cursors in die linke obere Ecke wird folgende Escape-Sequenz abgesetzt:

```
printf("\033[2J\033H");
```

Positionieren des Cursors / gotoxy

Setzen der nächsten Ausgabeposition (Cursor-gotoxy), erste Zahlenangabe entspricht der Y-Textkoordinate, zweite Zahlenangabe der X-Textkoordinate:

```
printf("\033[%d;%dH", y, x);
```

Grafik

Die Grafik-Sequenzen sind "eigentlich" nur als Gadget und als "nice to have" gedacht gewesen, könnten jedoch bei manchen Programmen zur Anwendung kommen.

Prinzipiell interpretiert die Grafik-Sequenz eine Abfolge von Befehlen für eine "Turtlegrafik". Hier wird ein unsichtbarer Grafikkursor positioniert, der mithilfe einer maximal 256 Byte großen Sequenz Linien, Kreise und Rechtecke (gefüllt und ungefüllt) in wählbaren Farben zeichnen kann. Hierfür wurde abweichend vom Standard Steuerzeichen 16 (0x10h) / DLE verwendet.

Dieser Sequenz folgen Anweisungen, wie eine Grafik zu zeichnen ist,

Beispiel

```
char draw_kreis[256];

strcpy(draw_kreis, "P64,30, c14, K20");
printf("%c%s\n", 0x10, draw_kreis);
```

Als erstes wird über den Platzhalter %c das Steuerzeichen 16 (0x10) gefolgt von der Grafikanweisung gesendet. Das Senden über einen Platzhalter ist deshalb notwendig, weil C hierfür kein Kürzel (wie bspw. \n oder \r für Linefeed oder Carriage return) kennt. Diesem Steuerzeichen folgt ein String, der die Anweisungen zum Zeichnen enthält. Dieser String darf nicht länger als 256 Byte (inklusive abschließendem Zerobyte) sein, da das Array, das die Zeichenanweisungen aufnimmt nicht mehr Elemente zur Verfügung stellt.

Im obigen Beispiel wird der Grafikkursor auf die Position x= 64, y= 30 gesetzt. c14 setzt die EGA-Farbe 14 und K20 zeichnet einen gefüllten Kreis mit der aktuellen Farbe (hier Farbe 14).

Zeichenanweisungen der Turtlegrafikfunktion		
Anweisung	Argument (e)	Funktion
P	x,y	Positioniert den Zeichenstift auf die absolute Koordinatenposition x,y ohne etwas zu zeichnen
p	x,y	zeichnet eine Linie zur angegebenen relativen (zur aktuellen) Koordinatenposition. '-' Zeichen kann vorangestellt werden
r	x	zeichnet eine Linie nach rechts mit den in x angegebenen Pixeln
l	x	zeichnet eine Linie nach links mit den in x angegebenen Pixeln
u	y	zeichnet eine Linie nach oben mit den in y angegebenen Pixeln
d	y	zeichnet eine Linie nach unten mit den in y angegebenen Pixeln
k	r	zeichnet einen Kreis mit dem in r angegebenen Radius in Pixeln
K	r	zeichnet einen gefüllten Kreis mit dem in r angegebenen Radius in Pixeln
t	b,h	zeichnet ein Rechteck von der aktuellen Zeichenposition mit der in b angegebenen Pixelbreite und in h angegebenen Pixelhöhe
T	b,h	zeichnet ein ausgefülltes Rechteck von der aktuellen Zeichenposition mit der in b angegebenen Pixelbreite und in h angegebenen Pixelhöhe
c	col	setzt die Zeichenfarbe col mit der in der EGA-Palette definierten Farbe
C	r,g,b	setzt die Zeichenfarbe, die mit (r)ot, (g)rün und (b)lau angegeben ist.

Die vordefinierten Farben der Turtlegrafik entsprechen NICHT den Farbnummern des Textterminals!! Die festgelegten Farben entstammen der Farbnummerierung der (ur)alten EGA Grafikkarte für damalige "IBM kompatiblen PC's) und gelten bis heute als die 16 Grundfarben für Textfarben im DOS-Modus.

Farbtabelle EGA-Palette			
0	schwarz	8	dunkelgrau
1	blau	9	hellblau
2	grün	10	hellgrün
3	türkis	11	helltürkis
4	rot	12	hellrot
5	lila	13	helllila
6	braun	14	gelb
7	hellgrau	15	weiss

Konfiguration des Displays

Mittlerweile sind eine Vielzahl von grafischen TFT-Displays erhältlich, die alle zueinander nur in mittlerweile geringerem Maße kompatibel sind. So gibt es bspw. Displays mit einer Auflösung von 128x128 Pixel sogar mit unterschiedlichen Displaycontrollern (hier dann mit ILI 9163, ST7735 oder ST7789) die optisch identisch aussehen, auch dieselbe Reihenfolge von Anschlusspins haben, aber dennoch unterschiedlich funktionieren.

Hier gab es dann sogar Displays, die intern gebondet sind, als ob es ein 160x128 Display wäre, was zu falscher Darstellung infolge von falscher Adressberechnung des Display-Rams führt. Dieses ist dann zu sehen, dass ein Teil des Displays nach der Initialisierung nur ein buntes Rauschen anzeigt. Zudem gibt es aber auch hier Displays, die einen kleinen Offset der Adressierung in X- und Y- Ausrichtung aufweisen, was sich mit einem buntem Rauschen am unteren und rechten Rand mit 2 bzw. 3 Pixel in Reihe und Spalte des Displays bemerkbar macht.

Bei anderen Displays ist die Adressierung des Display-Rams spiegelverkehrt, sodass eine Schrift eben spiegelverkehrt angezeigt wird.

Zu guter letzt gibt es auch Displays, die anstelle der "üblichen" Farbreihenfolge rot, grün, blau die Farbenfolge blau, grün, rot erwartet. Ist hier die falsche Farbenfolge eingestellt, ist das Erscheinungsbild farbverfremdet. Die Konfiguration des Displays erfolgt in der Datei **tft_termdisplay_v2.h**

Die Firmware von PicoMon setzt sich aus mehreren Dateien zusammen:

- Makefile
 - compiliert und linkt alle benötigten Programmteile zu einer flashbaren Binärdatei
 - Hinweis: Im Makefile für STM32F0xx kann/muß der Controller korrekt gewählt sein, um ggf. den erhöhten RAM-Bedarf des Programms bedienen zu können, für LSCRIPT steht zur Verfügung:

stm32f030x4.ld , stm32f030x6.ld , stm32f030x8.ld , stm32f070x6.ld , stm32f042x6.ld

- picomon.c
 - Quellcode und Main-Programm der Firmware
- spi_tft.c
 - Quellcode zur Ansteuerung der internen Hardware-SPI, sendet Daten vom Controller zum Display
- spi_tft.h
 - die dazugehörige Header-Datei
- tft_fonts.fnt
 - beinhaltet in C-Array Fonts in der Pixelgröße 5x7, 8x8, 10x16 und 12x16. Der Zeichensatz umfasst die Ascii-Zeichen ab Nummer 32 (das Leerzeichen) und reichen bis Nummer 255. Ascii-Zeichen von 128 .. 255 sind die Zeichen der ANSI-Codepage 850
- turtlegraph.c
 - beinhaltet den Interpreter zum Zeichnen von Turtlegrafiken
- turtlegraph.h
 - die dazugehörige Header-Datei
- my_printf.c
 - eine abgespeckte Variante von ' printf '
- my_printf.h
 - die dazugehörige Header-Datei
- uart.c
 - Quellcode zur Ansteuerung des internen USART, empfängt serielle Daten im 8N1 Protokoll mit 4800 Baud
- uart.h
 - die dazugehörige Header-Datei
- tft_term_modes.h
 - beinhaltet Deklarationen von Displayauflösung zu verwendetem Zeichensatz und Displayausrichtung, siehe Tabelle unten

- `tft_termdisplay_v2.c`
 - grundlegende Funktionen zum Betrieb des Displays, Initialisierung des Displays, Ausgabe eines einzelnen Bildpunktes sowie das Setzen von Textzeichen auf dem Display.
- `tft_termdisplay_v2.h`
 - die dazugehörige Header-Datei. Hier werden alle Einstellungen für den Betrieb des Displays vorgenommen:
 - Wahl des verwendeten Grafikcontrollers im Display
 - über die Angabe des 'terminal_mode' die Pixeldimensionen des Displays, den Zeichensatz und die Ausrichtung der Textausgabe auf dem Display
 - Einstellungen für unterschiedlichen Varianten der 128x128 Displays
 - gespiegelte Ausgabe
 - Farbreihenfolge

tft_termdisplay_v2.h

Innerhalb der Datei `tft_termdisplay_v2.h` gibt es eine Reihe von Einstellmöglichkeiten über `#define` - Schalter, um das Display konfigurieren zu können.

Die wichtigste ist hier ist die Hardwareanbindung des Displays an den Controller.

Gültig für CH32V003:

CH32V003F4P6									
+-----+									
PD4 / a7		1		20		PD3 / a4			
PD5 / a5 / utx		2	C	19		PD2 / a3			
PD6 / a6 / urx		3	H	18		PD1 / swio			
nrst / PD7		4	3	17		PC7 / miso			
PA1 / osc0 / a0		5	2	16		PC6 / mosi			
PA2 / osc1 / a1		6	V	15		PC5 / sck / scl			
gnd		7	0	14		PC4 / a2			
PD0		8	0	13		PC3			
Vdd		9	3	12		PC2 / scl			
PC0		10		11		PC1 / sda			
+-----+									
CH32V003				Display					

PC6-mosi	16	---->	sda / sdi / A0 / din						
PC5-sck	15	---->	sck						
PC4	14	---->	dc						
PC3	13	---->	ce / cs (kann auch direkt auf GND gelegt werden)						
PC2	12	---->	rst						

Da das Display mittels Hardware-SPI angebunden wird, können die Anschlüsse für `sda` und `sck` nicht gewählt werden

Der serielle Dateninput ist PD6, Pin 3

#define terminal_mode

terminal_mode wählt aus, für welche Displaydimensionen das Programm erstellt werden soll. Dieses erübrigt jedoch NICHT die obige Konfiguration des Displays bzgl. des Controllertyps, der Eigenheiten eines Displays wie etwa Farbreihenfolge oder Korrektur der Adressierung des Display-RAM

- Hinweis:
 - Modi mit RAM-Bedarf 2 kByte funktionieren auf Controllern CH32V003 und allen STM32F0xx
 - Modi mit RAM-Bedarf 4 kByte funktionieren alle STM32F030 Controller
 - Modi mit RAM-Bedarf 6 kByte funktionieren auf STM32F030c8 sowie alle STM32F042, F070 und F072

Gültige Werte für terminal_mode sind:

Mode	Pixel TFT	Font	Ausrichtung	Textzeichen	RAM-Bedarf
1	128x128	5x7	-	21x16	2k
2	128x128	8x8	-	16x16	2k
3	128x128	10x16	-	12x8	2k
4	128x128	12x16	-	10x8	2k
10	160x128	5x7	horizontal	26x16	2k
11	160x128	5x7	vertikal	21x20	2k
12	160x128	8x8	horizontal	20x16	2k
13	160x128	8x8	vertikal	16x20	2k
14	160x128	10x16	horizontal	16x8	2k
15	160x128	10x16	vertikal	12x10	2k
16	160x128	12x16	horizontal	13x8	2k
17	160x128	12x16	vertikal	10x10	2k
20	320x240	5x7	horizontal	53x30	6k
21	320x240	5x7	vertikal	40x40	6k
22	320x240	8x8	horizontal	40x30	4k
23	320x240	8x8	vertikal	30x40	4k
24	320x240	10x16	horizontal	32x15	2k
25	320x240	10x16	vertikal	24x20	2k
26	320x240	12x16	horizontal	26x15	2k
27	320x240	12x16	vertikal	20x20	2k
30	160x80	5x7	horizontal	26x10	2k
31	160x80	8x8	horizontal	20x10	2k
32	160x80	10x16	horizontal	16x5	2k
33	160x80	12x16	horizontal	13x5	2k

Beispiel:

```
#define terminal_mode 24
```

wählt ein Display 320x240 in waagerechter Ausrichtung mit einer Zeichenzahl von 32*15 Textzeichen aus.

Funktionsweise der Firmware

Die Funktionsweise der Firmware ist relativ einfach, auch wenn viele Dateien daran beteiligt sind. Der Controller wird mit Hardware-SPI für das Display und mit Hardware-USART für den Dateneingang initialisiert.

Der USART wird als Interrupteingang konfiguriert und die eingehenden Zeichen in einem Ringbuffer gespeichert. Dieses ist notwendig, damit bei einem eventuellen Scrollen des Displays keine eingehenden Zeichen verloren gehen.

Jedes eingegangene Zeichen wird gescant (parsing) und in einem Framebuffer abgelegt, der somit ein Abbild des dargestellten Displayinhalts hat sowie auf dem Display dargestellt. Dieser Buffer ist notwendig, damit ein Scrollen bei einem Linefeed in der untersten Reihe möglich ist.