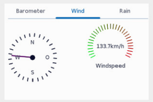
ANZEIGE
electronica fast forward the startup platform powered by elektor
COMPETE TO LAUNCH YOUR STARTUP AT ELECTRONICA 2020

Software

Von **Elektor LABS**
Software
Artikel

GUI mit Touch - für ESP32, Raspi und Co.

30. Dezember 2019, 10:00 Uhr



GUI mit Touch - für ESP32, Raspi und Co.

Grafische Benutzerschnittstellen erfordern häufig viel mehr Programmieraufwand als der „eigentliche Code“, der die wichtigen Funktionen des Projekts erledigt. Um sich daher auf die wesentlichen Aspekte des Projekts konzentrieren zu können, werden Fertiglösungen bzw. der Einsatz fertiger Bibliotheken für die grafische Ausgabe von Informationen immer beliebter und wichtiger. Bekannte Namen sind *µGFX*, *emWin* oder *TouchGFX* für STM32-Boards, um nur ein paar zu nennen. Alle haben Vor- und Nachteile, etwa bezüglich kommerzieller Lizenzierung oder der Bindung an bestimmte Controller-Hersteller. Natürlich könnte man auch selbst eine Bibliothek entwickeln, doch das ist eine Riesenmenge an Arbeit und bietet viele Fallstricke, von den vielen Bugs im selbstgestrickten, umfangreichen Code ganz zu schweigen.

Besser sieht es z.B. bei ***LittlevGL*** von Gábor Kiss-Vámosi aus, denn diese Bibliothek kommt mit einer sehr projektfreundlichen MIT-Lizenz daher. Ein damit entwickeltes GUI eignet sich gut für Touchscreens, kann aber auch mit Maus, Tastatur oder einigen Tastern bedient werden. Der Code läuft auf gängigen 16-, 32- und 64-Bit-Mikrocontrollern. Minimalkriterien sind 16 MHz, 64 KB Flash und 16 KB RAM. Damit passt die Bibliothek perfekt zu kleinen Boards wie dem ESP32 oder ESP8266 und wurde inzwischen von Espressif auch in ihr IDF aufgenommen. Nachfolgend gibt es Unterstützung für den Einstieg und die Zusammenstellung von Test-Hardware. LittlevGL bietet übrigens auch die Möglichkeit zur Entwicklung und Test von GUIs am PC, was nicht zu verachten ist. Den am PC erstellten Code für ein GUI kann man ohne große Anpassungen auf den Ziel-Mikrocontroller übertragen.

Bibliotheken & ESP32

Man lernt am meisten, wenn man etwas praktisch ausprobiert. Von daher wird hier der Einsatz der Bibliothek bei **der Wetterstation von Elektor** demonstriert. Ziel ist eine für die Touch-Bedienung geeignete GUI. Sogar eine mehrseitige Anzeige für Daten werden wir realisieren. Doch dazu bedarf es Hardware.

Ein ESP32-Modul ist leicht erhältlich. Geeignet sind z.B. ein *ESP32-PICO-D4*, ein *ESP32-DevKitC* oder ein davon abgeleitetes Board. Beim Display hat man die Wahl zwischen einem seriellen Interface oder paralleler Ansteuerung, die dann aber fast alle IOs des ESP32 verwendet. Da auch der Preis eine Rolle spielt, kommt hier ein verbreitetes 3,5“-LCD für den Raspberry Pi zum Einsatz. Die meisten günstigen Displays wie das 3,5“-Exemplar von JOY-iT werden per SPI angeschlossen und arbeiten mit 3,3-V-Pegeln auf den Signalleitungen. Sie eignen sich also perfekt für den Pins sparenden Anschluss an ein EPS32-Board. Zudem haben sie schon einen per SPI anschließbaren Touch-Controller integriert.

Die für RPi gedachten SPI-Displays sind allerdings in der Geschwindigkeit begrenzt, mit der Daten an das Display transferiert werden können. Wer so ein Display schon einmal an einem RPi in Aktion erlebt hat, hat die etwas träge Bildschirmaktualisierung schon zu spüren bekommen.

Wichtig: *LittlevGL* stellt keine Display-Treiber bereit, sondern nur „höhere“ Funktionen für das Zeichnen von Objekten. Es ist Sache des Entwicklers, die passenden hardwarenahen Routinen zu entwickeln. Aber auch dafür muss man das Rad nicht neu erfinden, denn für die meisten Display-Controller gibt es bereits fertige Bibliotheken. **Hier** wird auf die Arduino-Bibliothek *TFT_eSPI* zurückgegriffen, die auch 3,5“-Displays für RPi unterstützt.

Hardware

An Hardware für den Nachbau (siehe **Bild 1**) braucht es:

- ESP32-DevKitC-32D oder ESP32-PICO-Kit V4
- 3,5“ Touch-Display für Raspberry Pi von JOY-iT
- Kleine Steckbretter & Drahtbrücken

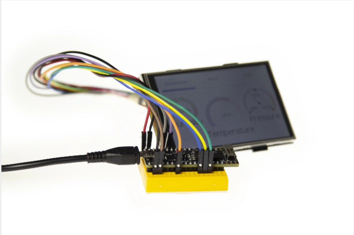


Bild 1. Minimales Setup.

Software

AKTUELLE AUSGABE

Aktuelle Ausgabe
2001EL
Inhalt anzeigen
Print-Version
Digital-Version (PDF)

KATEGORIEN

Info & Grundlagen
Bauteile
Advertorial
Elektor
Messtechnik
Mikrocontroller
Hard- & Software
Autoelektronik
Internet of Things
Embedded
Audio & Video
Stromversorgung
Elektor-Shop
Haus & Hof
Hobby & Modellbau
Tools & Printing
HF-Technik
Robotik
Sensoren
Platinenentwurf
Wearables
LEDs
Drohnen
Software

ANZEIGE

elektor pcb4makers
VISUALIZE & ORDER YOUR PCB. INSTANTLY.

MEISTGELESEN

15. Januar 2020
OpenMV Cam – intelligentes Auge für Embedded-Projekte
TEILEN / MEHR...

10. Januar 2020
Gratis-Artikel: DDS-Funktionsgenerator
TEILEN / MEHR...

24. Januar 2020
Gratis-Artikel: Card Sound
TEILEN / MEHR...

13. Januar 2020
Bzigo auf der CES: Tod den Stechmücken!
TEILEN / MEHR...

31. Januar 2020
Gratis-Artikel: Optischer Tastkopf für Oszilloskope
TEILEN / MEHR...

ANZEIGE

Learn & Design Electronics
Buch/Hardware-Bundles
ESP32
SDR
Elektor ESP32 Smart Kit Bundle
Elektor SDR Hands-on Kit
und andere
Zu den Elektor-Bundles >>

ANZEIGE
Raspberry Pi
APPROVED RESELLER

Auch die erforderliche Software ist übersichtlich. Neben der obligatorischen Arduino-IDE samt Board-Unterstützung für den ESP32 benötigt man noch die Arduino-Versionen der Bibliotheken *LittlevGL* und *TFT_eSPI*.

Um beide Bibliotheken komfortabel zu installieren und zu verwalten, müssen die beiden folgenden Adresszeilen in der Arduino-IDE unter *Preferences -> Additional Boards Manager URLs* eingetragen werden:

```
https://github.com/littlevgl/lv_arduino/library.json

https://github.com/Bodmer/TFT_eSPI/library.json
```

Damit sucht und installiert man *LittlevGL* und *TFT_eSPI* per Library-Manager. Anschließend prüft man, ob im Arduino-Bibliotheksordner „TFT_eSPI" und „LittlevGL" vorhanden sind.

Wie schon erwähnt braucht es beide Bibliotheken. *LittlevGL* kümmert sich um das UI, also die Animation und Anordnung von Objekten, die Verwaltung mehreren Szenen und das Rendern der angezeigten Grafiken. Resultat ist eine Bitmap. Diese Daten werden dann von *TFT_eSPI* in passender Weise an die Display-Hardware transferiert. Diese Bibliotheken abstrahieren daher vom konkret verwendeten Display.

Mehr Displays

TFT_eSPI unterstützt nicht nur SPI-Displays für RPi, sondern auch noch weitere mit folgenden Controllern: ILI9341, ST7735, ILI9163, S6D02A1, HX8357D, ILI9481, ILI9486, ILI9488, ST7789 und R61581.

Damit werden viele gängige Farb-Displays unterstützt. Sollte ein RAiO-basierter Controller wie etwa der RA8875 verwendet werden, so kann man stattdessen die ***RA8875-Library*** von Adafruit verwenden. Hierbei sind Anpassungen nötig, um *LittlevGL* anzubinden. Neben farbigen Displays können in Verbindung mit der ***u8g2-Library*** auch monochrome Typen verwendet werden. Der folgende Text bezieht sich allerdings auf die gebräuchlichen 3,5"-SPI-LCDs für RPi.

Eigener Treiber

Wenn ein Display mit einem Controller verwendet wird, für den es keinen Arduino-Treiber gibt, muss man wissen, welche Funktionen bereitgestellt werden müssen. Auch für eine Portierung und die Anbindung ist dieses Wissen hilfreich.

Prinzipiell reicht es, einen eigenen Treiber zu schreiben, der einzelne Pixel auf dem Display in einer definierten Farbe setzen kann. *LittlvGL* erwartet eine Funktion wie folgt:

```
/****************************************

* Function    : disp_flush_data

* Description : Sends pixels to the display

* Input       : lv_disp_drv_t *disp, int32_t x1, int32_t y1, int32_t x2, int32_t y2, const lv_color_t
*color_array

* Output      : none

* Remarks     : none

****************************************/

void disp_flush_data(lv_disp_drv_t *disp,
const lv_area_t *area, lv_color_t *color_p){

  /* Here: grab the pixel and send it to the display */



  /* Inform the library after job is done */

  lv_flush_ready(disp);

}
```

Diese Funktion wird *LittlevGL* als Funktions-Pointer übergeben. Als Parameter erhält sie die Start- und Endkoordinaten des zu füllenden Zeichenbereichs sowie einen Pointer auf die Bilddaten. Das konkrete Setzen der Pixel hängt vom Treiber für das jeweilige Display ab. Es kann allerdings vorkommen, dass die von *LittlevGL* gewünschten Farben noch für das Display umgerechnet werden müssen (z.B. von RGB nach BGR). Mehr braucht die eigentlichen Zeichenroutine nicht zu können. Andererseits gibt es Lösungen für eine Hardwarebeschleunigung, wie etwa durch die DMA2D-Engine einiger STM32-Controller.

Touchy

Damit wäre klar, wie eine Grafik auf das Display kommt. Was fehlt ist die Verarbeitung der Daten vom Touch-Controller. Hierfür gibt es eine LittlevGL-Funktion, welche die Koordinaten eines eventuellen Touchs geräteabhängig ausliest und verarbeitet. RPi-Displays sind in der Regel mit einem Touch-Controller des Typs *XPT2046* bestückt, der als weiterer Slave am SPI-Bus angebunden ist. Leider kann dieser nicht mit einem höheren Takt als 2,5 MHz ausgelesen werden. Da das Display und sein Controller mit einem Takt von 20 MHz (bei Raumtemperatur bis zu 26 MHz) laufen, muss bei einem Zugriff der Bus-Takt angepasst und anschließend wieder auf den ursprünglichen Takt zurückgesetzt werden. Auch hier hilft die Bibliothek *TFT_eSPI*, denn sie bietet nicht nur XPT2064-Unterstützung, sondern kümmert sich automatisch um die erforderliche Takt-Umschaltung.

Wer keine Touch-Bedienung nutzt, kann das UI auch mit Maus, Tastatur, Drehgebern oder Tastern bedienen. Selbstverständlich benötigen auch diese Eingabemethoden passende Treiber. Sie müssen in *LittlevGL* passend registriert werden.

Bildaufbau

Zunächst ein Wort zu den Stärken von LittlevGL: Vorteilhaft ist, dass auch der Quelltext zur Verfügung steht. Dies erleichtert das Debugging des eigenen Codes. Außerdem ist die Bibliothek gut dokumentiert und wird aktiv weiter entwickelt. Auch Einsteiger können mit den Beispielen schnell erste Erfolgserlebnisse erzielen und Oberflächen gestalten. Angefangen von einfachen Labeln über Buttons und Tabellen bis hin zu Dropdown-Listen und Gauges wird eine weite Palette an Bedienelementen bereitgestellt. Weiter werden Fenster und Hinweisboxen sowie Themes für das Erscheinungsbild geboten, um die GUI noch besser an eigene Bedürfnisse anpassen zu können. Die Dokumentation beschreibt alle Elemente detailliert. **Hier** werden auch Funktionen der Bibliothek und ihr Zusammenspiel demonstriert. *LittlevGL* bietet aktuell (noch) keine Basis-Funktionen zum Setzen von Pixeln oder Zeichnen von Linien. Der Grund liegt in der Art der Bildgenerierung: Wenn sich ein Element ändert, kann der neu zu zeichnende Bildschirmbereich ermittelt und der Puffer im internen RAM vorbereitet werden. Anschließend wird das Bild dann an das Display gesendet. Folglich müssen nicht nur eine Linie als Objekt vorliegen, sondern sogar einzelne Pixel. Diese Einschränkung erleichtert dafür das Neuzeichnen von Bereichen auf dem Bildschirm deutlich.

Nun zu den technischen Details der Generierung und Anzeige von Bildern: Wenn man flimmer- und störungsfreie Animationen oder Bildschirm-Updates haben möchte, könnte man zunächst das komplette Bild im RAM des Controllers vorbereiten und diese Daten anschließend erst ans Display übertragen. In unserem Fall hätte man es hier mit 307 KB Daten zu tun. Man könnte aber auch direkt alle Elemente an das Display übertragen und so weniger RAM belegen. Letzteres erschwert eine flickerfreie Anzeige und behindert Effekte wie Antialiasing, Transparenz und Schatten.

Ein Kompromiss ist das Spiegeln eines Bildschirmteilbereichs im RAM. Mit nur etwas mehr als 10 % des Speicherbedarfs für das Gesamtbild bekommt man schon alle angeführten Features. Für ein Display mit 480 x 320 Pixeln bei 16 Bit Farbtiefe wären dies „nur“ 30,7 KB RAM – für einen ESP32 ist das zwar eine Menge, die er aber durchaus packt.

In der aktuellen Version 6 der Bibliothek wird der Speicherbereich nicht durch ein #define mitgeteilt, sondern ist per Code bereitzustellen. Vor allem ist dieses Vorgehen praktisch, wenn weiteres externes RAM vorhanden ist und genutzt werden soll.

Bei unserer Demo beschränken wir uns auf eine statische Allokation eines Bereichs im Speicher des ESP32, was den Code einfach hält:

```
//Memory for the displaybuffer

static lv_disp_buf_t disp_buf;

static lv_color_t buf[LV_HOR_RES_MAX * 10];
```

Dieser Speicher wird dem Display in der Funktion hal_init() mit der folgenden Zeile zugewiesen:

```
lv_disp_buf_init(&disp_buf, buf, NULL, LV_HOR_RES_MAX * 10);
```

Bei anderen Mikrocontrollern muss überlegt werden, was möglich ist, denn etliche Exemplare haben deutlich weniger RAM zur Verfügung oder müssten über Klimmzüge externes RAM ansprechen. Neben dem verfügbaren RAM sind auch andere Aspekte wie die verfügbare Rechenleistung oder ein Multi-Threading relevant. *LittlevGL* kann leider kein Multi-Threading, weshalb alle Zugriffe aus demselben Thread erfolgen müssen, in dem auch die Funktion `lv_task_handler()` aufgerufen wird. Die nötige Rechenleistung hängt davon ab, wieviel Interaktion und Zeichnerei auf dem Display geschieht sowie ob und wie Animationen eingesetzt werden. Ein ESP32 hat dank seiner zwei Prozessorkerne genug Rechenleistung für ein GUI.

Experimente

Wer nun selbst experimentieren möchte, auf den lauern gelegentlich Fallstricke. Für eine reibungsarme Inbetriebnahme wird nachfolgend ein Setup beispielhaft beschrieben. Ein ESP32-D4-PICO-Board hat durch die zusätzliche Last eines Displays gelegentlich leichte Startprobleme. Ein zusätzlicher Kondensator von 10 µF zwischen 3,3 V und GND verzögert das Booten soweit, bis die Spannungen in einem definierten Bereich sind.

Der Anschluss eines Displays an ein ESP32-Board erfolgt anhand der Belegung von **Tabelle 1**. Damit wäre die Hardware vorbereitet. Nun folgen die Konfiguration und der Test der Software. Zunächst geht es um die Bibliothek *TFT_eSPI* als eigentlichem Display-Treiber und anschließend um die Konfiguration von *LittlevGL*.

Tabelle 1. ESP32-Verdrahtung.

Funktion	Display-Pin	ESP32-Pin	Bemerkung
MISO	21	19	
MOSI	19	23	
SCK	23	18	
DC	18	02	
CS	24	05	
RST	22	EN	spart 1 GPIO-Pin ein
T_CS	26	04	
VCC (5 V)	02	5 V	
GND	14 / 25	GND	
T_IRQ (optional)	11	34	ESP32-Pin nur Input

Beim Display-Treiber muss man im Bibliotheksverzeichnis der Arduino-IDE den Ordner *TFT_eSPI* suchen, um die Datei *User_Setup.h* an das genutzte Display anzupassen. Für das verwendete Display müssen die folgenden `#defines` vorhanden sein:

```
#define RPI_ILI9486_DRIVER // max. 20 MHz SPI

#define TFT_MISO 19

#define TFT_MOSI 23

#define TFT_SCLK 18

#define TFT_CS   05 // Chip select control

#define TFT_DC   02 // Data Command control

#define TFT_RST  -1 // set TFT_RST to -1 if display RESET is connected to ESP32 RST

#define TOUCH_CS 04 // Chip Select (T_CS) of touch screen

#define SPI_FREQUENCY  20000000


// An XPT2046 requires a lower SPI clock rate of 2.5 MHz:
```

```
#define SPI_TOUCH_FREQUENCY 2500000
```

Es werden also die verwendeten GPIOs definiert und ein 20-MHz-SPI-Takt dient als sicherer Anfangswert für das Display. 2,5 MHz eignen sich für den Touch-Controller. Zum Testen wählen wir ein Beispiel (Auswahl: *TFT_eSPI -> 480x320 -> Rainbow480*), das einmal die Regenbogenfarben ausgibt. Wenn alles Kompiliert und richtig angeschlossen ist, sollte das Display wie in **Bild 2** aussehen. Damit ist die Hardware grundsätzlich einsatzbereit.

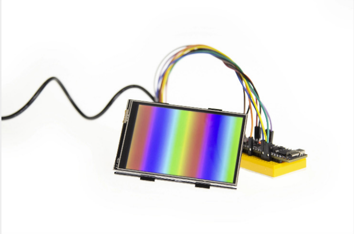


Bild 2. Erster Test: Das Display zeigt Regenbogenfarben.

Als nächster Schritt erfolgt die Anbindung von *LittlevGL* an den Display-Treiber und die Erstellung eines eigenen HMI (Human Maschine Interface). Zur Nutzung von *LittlevGl* muss man zuerst die Konfiguration anpassen. Hierzu sucht man im Arduino-Bibliotheksverzeichnis den Ordner *littlevGL*. In der dortigen Datei *lv_config.h* werden Anpassungen an das verwendete Display vorgenommen und die verfügbaren Elemente der Bibliothek eingestellt. Am Anfang der Datei befinden sich die Einstellungen für die Speicherverwaltung.

Die Zeile:

```
#define LV_MEM_SIZE (64U * 1024U)
```

definiert das für Objekte der GUI reservierte RAM. Beim angegebenen Wert von 64 kB wird später der Linker bemängeln, dass so viel nicht bereit gestellt werden kann. Bei statischem Reservieren (zur Compile-Zeit) macht sich der nicht durchgehende Speicherbereich des ESP32 bemerkbar. Man könnte nun zur Laufzeit via `malloc` und `free` entsprechende Blöcke reservieren. Da diese Maßnahme andere Gefahren birgt, wird die Sache anders angegangen. Man ändere hierzu die Zeile in:

```
#define LV_MEM_SIZE (24U * 1024U)
```

Dies ist für unsere ersten Schritte ausreichend.

Das Display hat eine Auflösung von 480 x 320 Pixel. Die zugehörigen `#defines` sind:

```
/* Horizontal and vertical resolution of the library */
#define LV_HOR_RES_MAX (480)
#define LV_VER_RES_MAX (320)
```

Die Auflösung in DPI (Dots Per Inch) ergibt sich nach folgender Formel:

$$DPI = \frac{\sqrt[2]{(horizontal\ resolution)^2 + (vertical\ resolution)^2}}{screen\ diagonal\ in\ inch}$$

Eingesetzt:

$$DPI = \frac{\sqrt[2]{(480)^2 + (320)^2}}{3{,}5} = 164{,}83$$

Ganzzahlig ergibt sich daraus:

```
#define LV_DPI 164
```

Soviel zur Basiseinstellung. Für erste Tests bleiben die restlichen Einstellungen unangetastet; die Änderungen werden gesichert. In der Arduino-IDE kann man nun unter *LittlevGL* das Beispiel *ESP32_TFT_eSPI* auswählen und auf das ESP32-Board laden. Ist alles richtig konfiguriert, sollte sich „Hallo Arduino!“ auf weißem Grund auf dem Display zeigen.

Treiber und *LittlvGL* kooperieren also gut. Allerdings wurden der Touch-Controller des Displays noch nicht ausgelesen und diese Daten nicht an die Bibliothek übergeben. Nachfolgend daher die grundlegenden Code-Teile, womit man ein Grundgerüst für eine eigene Anwendung erstellen kann. Dazu wird das Beispiel *ESP32_TFT_eSPI* aus der LittlevGL-Bibliothek, das gerade in den ESP32 geladen wurde, näher angeschaut.

In der Funktion `setup()` folgen nach der Initialisierung der Bibliothek in Zeile 63 mit `lv_init()` und des TFTs in den Zeilen 69 und 70 mit `tft.begin()` und `tft.setRotation(1)` schließlich die Zeilen 73 und 74 mit der Initialisierung des Structs `lv_disp_drv_t`. Diesem Struct wird ein Funktions-Pointer für das Schreiben auf das Display mitgegeben und danach in der Bibliothek registriert.

Ein ähnliches Vorgehen findet man beim Dummy-Touch-Treiber in den Zeilen 80…84. Als letzter Schritt wird mit Hilfe eines „Tickers" eine Zeitbasis für die Bibliothek bereitgestellt. Hierbei wird eine Funktion im vorgegebenen Intervall von 20 ms aufgerufen. Jedes Mal wird ein Zeitzähler um 20 ms erhöht. Abschließend wird dann eine Schaltfläche angelegt und dieser der Text „Hallo Arduino!" zugewiesen (Zeilen 90…92).

Innerhalb der Loop-Funktion wird nun noch `lv_task_handler()` aufgerufen, damit die GUI auf Eingaben reagieren oder den Bildschirm neu zeichnen kann.

Damit man nicht bei jedem Projekt wieder bei Null anfangen muss, hat der Autor ein Basis-Projekt erstellt, in dem die Einstellungen für das Display von JOY-iT und seinen Touch-Controller sowie die Initialisierung der Komponenten vorgenommen werden. In Zeile 139 des Sketches wird die Orientierung des Displays mit `tft.setRotation(3)` eingestellt. Das Bild wird so um 270° gegenüber der Ausgangsposition gedreht. Braucht ein anderes Display z.B. eine Drehung um 180°, muss der Parameter auf 1 gesetzt werden.

GUI-Erstellung

Mit diesem Grundgerüst kann man anfangen, ein eigenes GUI zu erstellen. Dies kann man direkt auf der ESP32-Hardware erledigen, doch das Kompilieren, Hochladen und Testen nimmt Zeit in Anspruch. Die Alternative ist ein PC-Simulator. Seine Installation setzt die Vertrautheit mit *Eclipse* voraus. Die Installation wird **hier** beschrieben. Sie ist unter Windows etwas schwieriger als unter Linux oder OS X. Im Simulator können nun die ersten Schritte ausprobiert werden, ohne jedes Mal modifizierten Code auf die Hardware laden zum müssen.

Zuerst geht es um das Design der Oberfläche, wozu man am besten zunächst zu Stift und Papier (oder zu Tablet und Stift) greift, denn vor den ersten Code-Zeilen sollten Skizzen angefertigt werden. **Bild 3** zeigt ein Beispiel solch einer Hand-Skizze. So wird klar, an welcher Stelle welches Objekt platziert und mit welchen Objekten navigiert wird.

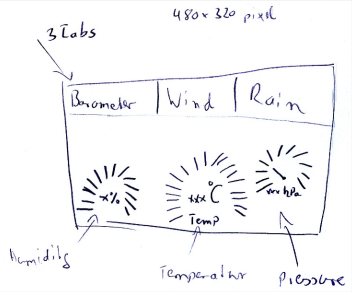


Bild 3. Beispiel einer Handskizze für die Anzeige von Luftdruck, Temperatur und Luftfeuchte mit drei Tabs.

Da es um eine Wetterstation geht, wird für die Anzeige der Wetterdaten eine einfache Oberfläche mit drei Tabs erstellt. Damit der Arduino-Sketch übersichtlich bleibt, werden die Funktionen und die Erstellung der GUI-Elemente in einer eigenen Datei abgelegt.

Zuerst wird die Szene vorbereitet und ein Tabview-Element angelegt, dem dann drei Tabs hinzugefügt werden: *Barometer*, *Wind* und *Rain*. Für die Vorbereitung der Szene ist der folgende Code zuständig:

```
lv_theme_set_current(th);


/* Next step: create a screen */

lv_obj_t * scr = lv_cont_create(NULL, NULL);

lv_scr_load(scr);
```

Zuerst wird das Theme geladen, das als Funktionsparameter übergeben wurde. Anschließend wird eine leere Szene angelegt und geladen. Das Tabview-Element bekommt die komplette Bildschirmgröße zugewiesen. Der Screenshot in **Bild 4** zeigt die drei leeren Tabs mit den durch das Theme vorgegebenen Schrifteinstellungen.

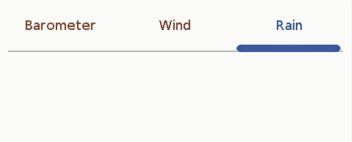


Bild 4. Bildschirm mit drei Tabs ohne weiterem Inhalt.

Klickt man auf den Namen des entsprechenden Tabs, wird ein Wechsel der Tabs durch den blauen Marker angezeigt. Da die Tabs leer sind, sieht man nicht mehr als dies.

Mit den folgenden fünf Code-Zeilen

```
/* And now populate the four tabs */

lv_obj_t * tv = lv_tabview_create(scr, NULL);

lv_obj_set_size(tv, LV_HOR_RES_MAX, LV_VER_RES_MAX);

lv_obj_t * tab0 = lv_tabview_add_tab(tv, "Barometer");

lv_obj_t * tab1 = lv_tabview_add_tab(tv, "Wind");

lv_obj_t * tab2 = lv_tabview_add_tab(tv, "Rain");
```

werden die ersten drei Tabs angelegt. Zunächst haben sie noch keinen Inhalt und werden daher anschließend mit Titeln versehen.

Wetter-Anzeige

Los geht es mit dem Barometer: Hier sollen drei Werte für Luftfeuchtigkeit, Temperatur und Luftdruck angezeigt werden. Für Luftfeuchtigkeit und Temperatur werden `lv_lmeter` und `label` verwendet, welche Wert und Namen der Messgröße anzeigen. Für die Luftfeuchtigkeit wird `lv_gauge` genutzt. Bequemerweise kann man das Aussehen der Elemente noch zur Laufzeit durch Styles beeinflussen und so jedes Element individualisieren.

Bei der Anordnung von Elementen muss auf die Autofit-Funktion der Bibliothek geachtet werden. Man muss also die Elemente passend anordnen oder Autofit abschalten. Elemente lassen sich von mehreren Ursprungskoordinaten aus positionieren – eine Übersicht findet sich **hier**. Die einzelnen Elemente können *Parents* besitzen, also Objekte, von denen ihre Position abhängt. Auf diese Weise ergeben sich elegante Abhängigkeiten der Positionen, wodurch sich bei einer Parent-Verschiebung auch alle *Childs* neu ausrichten. Im Code kann auf die angelegten Elemente später nicht mehr direkt zugegriffen werden. Für *LMeter* und *Gauges* nutzen wir daher Pointer, auf die man global zugreifen kann. Im Beispiel-Code

```
lv_obj_t* humidity_lmeter

lv_obj_t* humidity_label

lv_obj_t* temp_lmeter

lv_obj_t* temp_label

lv_obj_t* air_pressure_gauge
```

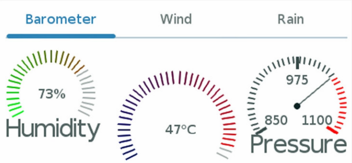
fällt auf, dass Funktionen wie `lv_lmeter_create` immer nur Pointer zurückliefern. Die Frage ist nun, wo Speicher reserviert wird. Dies ist etwas tiefer in der Bibliothek vergraben. Der Ausdruck:

```
# define LV_MEM_SIZE (24U * 1024U)
```

legt einen festen Speicher-Pool für die Grafikelemente an. Bei jedem Aufruf einer Create-Funktion wird etwas Speicher aus dem Pool genommen und für das Grafikobjekt reserviert. Das Ergebnis der Operation ist ein Pointer auf die Speicheradresse, der dazu genutzt wird, Eigenschaften des Objekts zu ändern. Falls irgendwann der Speicher ausgeht – bei dynamischen Oberflächen kann das passieren – wird in der Bibliothek ein Fehler ausgelöst und in eine Endlosschleife verzweigt. Der ESP32 stoppt komplett.

Die Pointer sind erst mal nur in der Funktion gültig, in der man sich gerade befindet. Will man später ohne Umwege auf ein Element zugreifen, so müssen die Pointer außerhalb der Funktion abgelegt werden. Der Einfachheit halber dienen hierzu bei uns ein paar globale Variablen. Bei ernsthaften Anwendungen ist von der Verwendung globaler Variablen aber abzuraten.

Via Pointer kann man dann neue Werte in die Anzeigen schreiben. Exemplarisch hierfür ist z.B. die Funktion `UpdateTemperature`. Beim Anzeige-Element *Lmeter* wird ein Wert zwischen 0 und 100 erwartet, doch die Größe hat einen Wertebereich von ±50°. Die Temperatur muss also mit einem Offset von 50 versehen werden. 0° entsprechen dann einem *Lmeter*-Wert von 50. Zusätzlich wird die aktuelle Temperatur noch als Text angezeigt. Dies wird durch `snprintf` und einen kleinen lokalen Puffer erledigt, der als neuer Text in das Textfeld geschrieben wird. Ändert sich die Textlänge, wird der Text nicht automatisch neu ausgerichtet. Die Ausrichtung muss nach dem Setzen des Textes erneut vorgenommen werden. Hierzu wird `lv_obj_align` mit den Parametern für das Label erneut aufgerufen. Luftfeuchtigkeit und Luftdruck werden ganz ähnlich behandelt. **Bild 5** zeigt einen Screenshot des fertigen Tabs und in **Bild 6** kann man sehen, wie das auf dem LCD „in echt" aussieht.



Temperature

Bild 5. Screenshot des 1. Tabs mit Barometerwerten.

Barometer
Wind
Rain
100%
Humidity
13°C
Temperature
975
850
1100
Pressure

Bild 6. Barometer auf realem Display.

Der erste Tab wurde nun mit Inhalten versehen. Beim zweiten Tab wird ähnlich verfahren, jedoch erfordert die Windrichtungsanzeige als Kompass etwas mehr Aufwand. Hier fungiert ein *Gauge* als *Parent* für vier *Label*. Im Code wird zunächst ein *Gauge* mit 0…359° angelegt. Darauf folgen vier *Label*, denen der Kompass als *Parent* verpasst wird. Die *Label* werden auf dessen Mitte bezogen definiert. Somit ergeben sich die vier Himmelsrichtungen. Der Zeiger weist in die Richtung, aus welcher der Wind kommt. Beim *Gauge* ergeben sich 0° nicht beim Wert 0, sondern bei 180. Zur Anzeige der Windgeschwindigkeit wird wieder ein *Lmeter* mit ähnlichem Aufbau wie beim Barometer genutzt. Man merkt, dass sich für das Anlegen der Elemente Vorgänge wiederholen. Zunächst wird ein Style für das Objekt angelegt, dann erfolgt das Erstellen des Objekts und zum Schluss werden ihm seine Eigenschaften zugewiesen. Das fertige Resultat ist im Screenshot von **Bild 7** zu sehen.

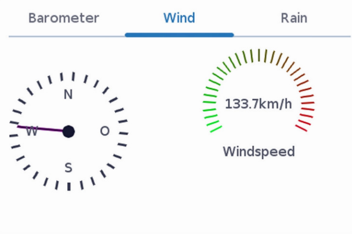


Bild 7. Tab mit Windrichtung und Geschwindigkeit.

Beim Regen bzw. Niederschlag sieht die Darstellung der Werte anders aus. Hier werden die Werte in Textform sowie in einem Diagramm dargestellt, aus dem sich der zeitliche Verlauf ergibt. Die Texte werden wie schon zuvor beschrieben realisiert: Erst werden Styles und Objekte angelegt – dann die Werte zugewiesen. Für den Verlauf der Regenmenge eignet sich ein Liniendiagram, welches seit der Version 6 auch keine Tricks zur Beschriftung der Achsen mehr benötigt. Zur Aktualisierung der Werte muss aber nicht jeder Datenpunkt einzeln bewegt werden, denn `lv_chart_set_next` erledigt diese Aufgabe. Einmal pro Stunde wird ein neuer Wert an das Diagramm übergeben. Die Aktualisierung der Niederschlagsmengen erfolgt wie bei anderen Texten auch durch eine eigene Funktion. **Bild 8** zeigt einen Screenshot mit Pseudodaten des Verlaufs und der Werte des Niederschlags.

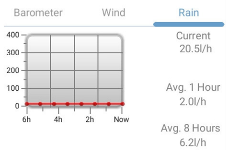


Bild 8. Screenshot des Tabs mit Niederschlagsverlauf und Werten.

Für die Datenanbindung des Displays wird **Code des Projekts Monster-LED-Uhr** recycelt, da hier schon Daten per MQTT von einem Broker kommend verarbeitet werden. Der Code erwartet, dass der Broker einen JSON-String sendet, in dem die Werte für Luftfeuchtigkeit, Temperatur, Luftdruck, Windrichtung, Windgeschwindigkeit und die Niederschlagsmengen enthalten sind. Wenn vom Broker neue Daten kommen, werden diese in die zugehörigen Elemente eingetragen. Man muss allerdings darauf achten, dass dies nicht durch unterschiedliche Threads passiert. Auch bei der Konfiguration gibt es bis auf die fehlende Uhrzeit-Einstellung keine großen Unterschiede zum **LED-Uhr-Projekt**. WLAN- und MQTT-Einstellungen wurden schlicht übernommen, es müssen lediglich die Wetterstation und die Anzeige auf das gleiche Topic eingestellt werden. Ab da gelangen die Werte direkt aufs Display. Eine Ausnahme ist im Moment noch die Regenmenge, denn hier wird nur die aktuelle Regenmenge der Wetterstation ausgegeben. Die Berechnung von Stundenwerten und dem Verlauf sind bei der Wetterstation leider noch nicht implementiert. Sobald dieses aber geschehen ist, werden auch diese Werte auf dem Display aktualisiert.

Fazit

Mit diesem Beispiel wurden einige grundlegenden Funktionen von *LittlevGL* praktisch demonstriert. Der zugehörige Code für diese Demonstration kann unten heruntergeladen werden. Wie schon gesagt, stellt *LittlevGL* aber noch deutlich mehr Funktionen und Animationen sowie Gestaltungsmöglichkeiten mit Tabellen, Listen und Dropdown-Menüs zur Verfügung.

Wenn Sie nun von der Einfachheit der Nutzung dieser Bibliothek angetan sind, werden Sie *LittlevGL* sicherlich auch in eigenen Projekten ausprobieren wollen. Ein EPS32-Modul in Verbindung mit einem Touch-Display ist ja schließlich eine recht universell einsetzbare Plattform, die viel Leistung für wenig Geld bietet.

Wollen Sie weitere Elektor-Artikel lesen? Jetzt Elektor-Mitglied werden und nichts verpassen!

DOWNLOAD

190295-11_GUI WITH TOUCH.ZIP

DATEI HERUNTERLADEN

Klicken Sie auf den Button, um den Einbettungscode zu erhalten.

Kommentar schreiben

KOMMENTARE

Holger Andrä 16. Januar 2020, 00:12 Uhr
Wo bitte gibt es den Artikel als PDF-Download?

KOMMENTARE ANZEIGEN

Manfred Häfner 14. Januar 2020, 10:34 Uhr
Bitte stellen Sie den Artikel auch als PDF zum Download bereit. Danke

KOMMENTARE ANZEIGEN

HaSch 6. Januar 2020, 14:17 Uhr
Liebes Elektor LABS Software Team,

bitte überprüft *dringend* noch einmal die Download Dateien, da ist einiges faul:
Leider wird das PDF des Artikels nicht angeboten, weshalb der Artikel als "Download" nutzlos ist.
Ich habe den Arduino-Sketch mit allen Libraries und Dateien runtergeladen und installiert. Das Kompilieren produziert einen Haufen Fehler, die alle mit der LittlevGL Library zusammenzuhängen scheinen. Ich habe alle Versionen der Library durchprobiert mit allen möglichen Einstellungen - es funktioniert leider nicht!!!

KOMMENTARE ANZEIGEN 8 Kommentare

HaSch 6. Januar 2020, 10:37 Uhr
Software doesn't compile! It produces a lot of errors regarding LittlevGL library. It doesn't matter which version of lvgl is used.

KOMMENTARE ANZEIGEN

HaSch 6. Januar 2020, 10:34 Uhr
Please add PDF file to the download!

KOMMENTARE ANZEIGEN

Alexander 3. Januar 2020, 15:28 Uhr
Irgendwie war ich immer der Meinung das so etwas wie Elektor eine Lern Zeitung ist. Bei Analoger Elektronik kommt die Elektor dem auch noch am nächsten. Aber die Projekte gehen eigentlich immer den Weg des geringsten Widerstands. Das ist dem Lerngedanken nicht zuträglich. Sicher solche Bibliotheken wie oben benutzt sind hilfreich und auch wichtig. Nur wenn jeder nur noch die Bibliotheken benutzt wird es nicht lange dauern bis man den Aufschrei hört FACHKRÄFTEMANGEL es gibt keine Leute mehr die wissen wie man Bildchen auf ein Display malt. Offensichtlich ist dem Westen daran gelegen in eine Technologische Abhängigkeit der Asiatischen Länder zu kommen.

KOMMENTARE ANZEIGEN 8 Kommentare

Tilo Laur 3. Januar 2020, 10:15 Uhr
Wo ist denn dieses Mal der Artikel als downloadbare PDF-Datei?

KOMMENTARE ANZEIGEN

Kommentar schreiben

« ZURÜCK TEILEN

VERWANDTE ARTIKEL

GitHub für Dummies

MEHR...

Gratis-Artikel: Netzausfall-Detektor mit SMS-Alarm

MEHR...

PEAK-System's Neuheiten rund um CAN FD und CANopen FD auf der embedded world 2020

MEHR...

Bauen Sie ein PM2.5-Feinstaub-Messgerät!

MEHR...

SERVICE

Datenschutzerklärung
Nutzungsbedingungen
Copyright
Mediadaten
Impressum

ELEKTOR WORLD

Elektor MAGAZINE
Elektor LABS
Elektor SHOP

FOLGEN SIE UNS

SICHER BEZAHLEN

VISA MasterCard PayPal

Newsletter abonnieren und Vorteile sichern