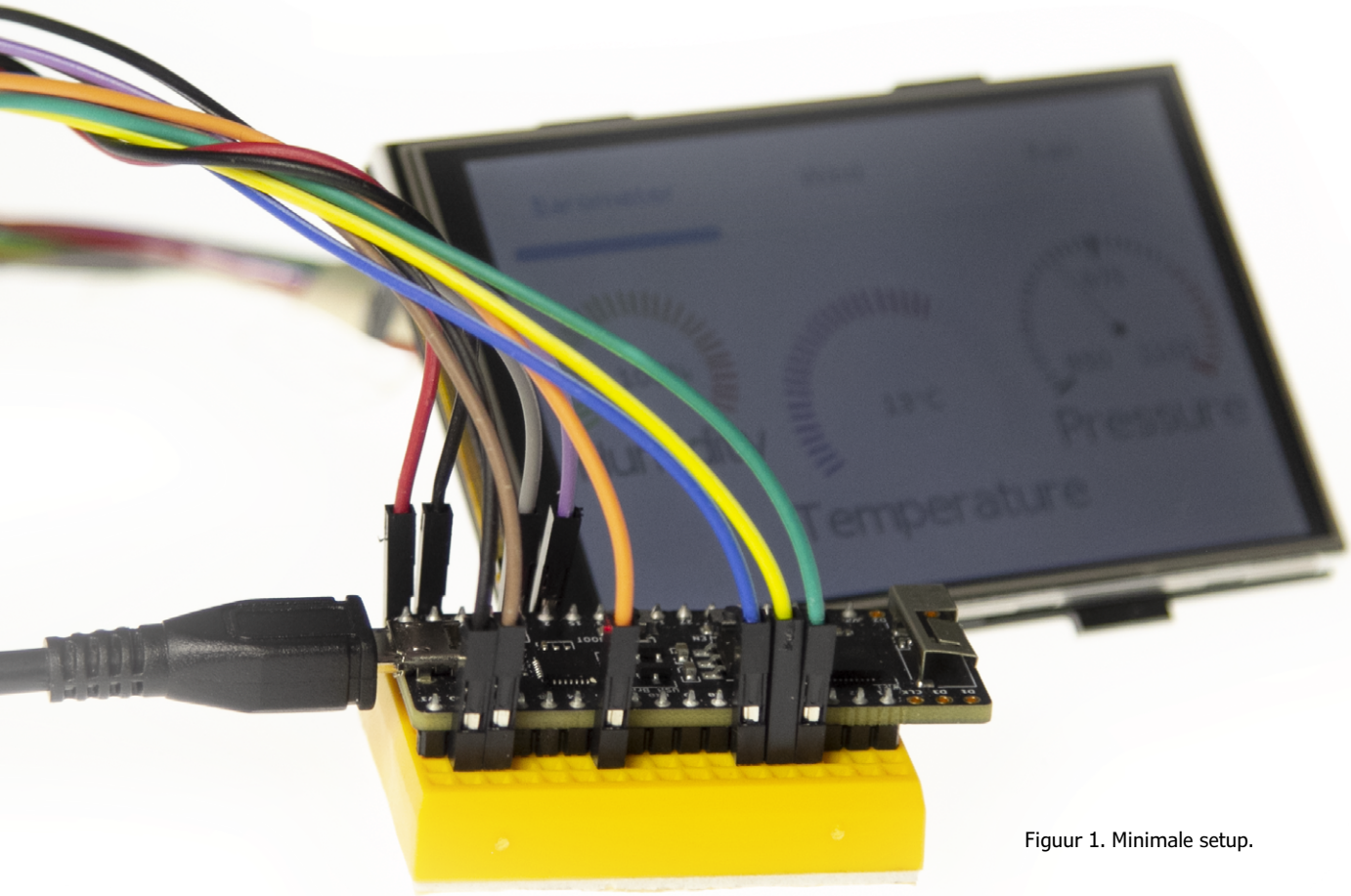
Touch-GUI voor ESP32 en Raspberry & co

grafische gebruikersinterface met de LittlevGL-bibliotheek

Mathias Claußen (Elektor Labs)

Bijna alle microcontrollerprojecten moeten iets weergeven. Waar men vroeger nog kon volstaan met tweeregelige tekstdisplays, ziet men tegenwoordig steeds meer grafische LC- of OLED-displays. En inmiddels is er dan ook een hele reeks software-libraries beschikbaar waarmee aantrekkelijke grafieken en/of touch-bediening gerealiseerd kunnen worden. LittlevGL is een library met een zeer ruime MIT-licentie, die u eenvoudig aan verschillende displays en controllers kunt aanpassen. In dit artikel laten we zien hoe dit in zijn werk gaat aan de hand van een ESP32-board met een touch-LCD dat de gegevens van een weerstation toont.



Figuur 1. Minimale setup.

De implementatie van een goede grafische gebruikersinterface kost vaak veel meer programmeerwerk dan de 'eigenlijke code' die de belangrijke functies van het project voor zijn rekening neemt. Om die reden worden kant-en-klare oplossingen of bibliotheken voor grafische uitvoer steeds populairder en belangrijker. Ze maken het de ontwikkelaar een stuk gemakkelijker, zodat die zich op de wezenlijke aspecten van het project kan concentreren. Bekende libraries zijn *µGFX*, *emWin* en *TouchGFX* voor STM32-boards, om er maar een paar te noemen. Ze hebben allemaal zo hun voor- en nadelen, zoals licentiëring voor commerciële toepassingen of binding aan specifieke controller-fabrikanten. In principe zou je ook zelf een library kunnen ontwikkelen, maar dat is enorm veel werk en er zijn nogal wat valkuilen, om nog maar te zwijgen van de hoeveelheid bugs in omvangrijke zelfgeschreven code.
Dan is bijvoorbeeld de bibliotheek *LittlevGL* van Gábor Kiss-Vámosi [1] veel aantrekkelijker. Voor het gebruik van deze bibliotheek geldt een heel projectvriendelijke MIT-licentie. Een GUI

ontwikkeld met LittlevGL is geschikt voor een aanraakscherm, maar kan ook worden bediend met muis, toetsenbord of een paar drukknoppen. De code draait op gangbare 16-, 32- en 64-bits-microcontrollers op minimaal 16 MHz en met 64 KB flash en 16 KB RAM of meer aan boord. Dat maakt deze library ideaal voor kleine boards zoals de ESP32 of ESP8266, en daarom heeft Espressif hem inmiddels ook in hun IDF opgenomen. Bovendien is er ondersteuning, zoals de nodige tutorials en hulp bij het samenstellen van test-hardware. Niet onbelangrijk is verder dat u met LittlevGL ook op uw PC GUI's kunt ontwikkelen. De resulterende code kan zonder omvangrijke aanpassingen naar de beoogde microcontroller worden geport.

Bibliotheken en de ESP32

Al doende leert men het meeste. Daarom laten we hier zien hoe deze bibliotheek kan worden gebruikt voor een weerstation van Elektor [2]. We willen een GUI voor een aanraakscherm, en voor de weergave van data gaan we meerdere pagina's gebruiken. Maar daarvoor hebben we wel eerst wat hardware nodig. ESP32-modules zijn gemakkelijk verkrijgbaar. Geschikt voor dit project zijn bijvoorbeeld de *ESP32-PICO-D4* of de *ESP32-DevKitC* of afgeleiden daarvan. Voor het display kunnen we kiezen tussen een seriële interface of parallelle aansturing, waarbij die laatste dan meteen bijna alle I/O's van de ESP32 bezet. De prijs speelt ook een rol, daarom kiezen we voor een bekend 3,5" LCD voor de Raspberry Pi. De meeste goedkopere displays, zoals het 3,5"-exemplaar van JOY-iT, worden aangestuurd via SPI en werken met een signaalniveau van 3,3 V. Dat maakt deze displays dus perfect geschikt voor aansluiting op een ESP32-board waarbij zo min mogelijk pinnen worden gebruikt. Bovendien beschikken ze over een geïntegreerde touch-controller, die eveneens via SPI aangesloten kan worden.
SPI-displays voor de Raspberry Pi zijn echter wel wat beperkt wat betreft de snelheid waarmee data naar het display kan worden overgedragen. Als u zo'n display al eens in actie hebt gezien in combinatie met de Raspberry, dan hebt u ongetwijfeld al bemerkt dat het beeldscherm vrij traag wordt ververst. Belangrijk: *LittlevGL* heeft geen display-drivers, maar biedt alleen 'hogere' functies voor het tekenen van objecten. De ontwikkelaar dient dus de routines voor de interactie met de hardware zelf te ontwikkelen. Maar ook hier hoeven we het wiel niet opnieuw uit te vinden, want voor de meeste display-controllers bestaan kant-en-klare libraries. Wij grijpen terug op de Arduino-library *TFT_eSPI* [3], die ook 3,5"-displays ondersteunt.

Hardware

Voor het nabouwen van dit project hebt u het volgende nodig (zie **figuur 1**):

- ESP32-DevKitC-32D of ESP32-PICO-Kit V4
- 3,5" touch-display voor de Raspberry Pi (JOY-iT)
- kleine breadboards en jumperdraden

Software

Wat u aan software nodig hebt is ook vrij overzichtelijk. Naast de verplichte Arduino-IDE met board-ondersteuning voor de ESP32 hebt u de Arduino-versies van de bibliotheken *LittlevGL* en *TFT_eSPI* nodig.
Om beide bibliotheken comfortabel te installeren en te gebruiken dient u beide onderstaande adresregels in te voeren in de Arduino-IDE, onder *Preferences → Additional Boards Manager URLs*:

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

Op deze manier zoekt en installeert u *LittlevGL* en *TFT_eSPI* met behulp van de Library Manager. Vervolgens gaat u na of 'TFT_eSPI' en 'LittlevGL' in de Arduino library-map aanwezig zijn.
Zoals al opgemerkt hebben we beide bibliotheken nodig. *LittlevGL* zorgt voor de UI, dus voor de animatie en de toewijzing van objecten, het beheer van meerdere scenes en het renderen van grafieken. Het resultaat is een bitmap. Deze data wordt dan met *TFT_eSPI* op gepaste wijze naar de display-hardware overgezet. De bibliotheken abstraheren in wezen het display dat u gebruikt.

Meer displays

TFT_eSPI ondersteunt niet alleen SPI-displays voor de Raspberry, maar ook andere met deze controllers: ILI9341, ST7735, ILI9163, S6D02A1, HX8357D, ILI9481, ILI9486, ILI9488, ST7789 en R61581.
Hiermee worden heel veel gangbare kleurendisplays ondersteund. Als u een RAiO-controller zoals de RA8875 gebruikt, dan kunt u de *RA8875-library* van Adafruit [4] gebruiken. Dan moet u wel het een en ander aanpassen om *LittlevGL* met deze library te koppelen. Naast kleurendisplays zijn in combinatie met de *u8g2-library* [5] ook monochrome types te gebruiken. De volgende tekst gaat echter over de gebruikelijke 3,5"-SPI-LCD's voor de Raspberry.

Eigen drivers

Als u een display gebruikt met een controller waarvoor geen Arduino-driver is, dan moet u weten welke functies klaargezet moeten worden. Dat is trouwens ook handig om te weten als u bestaande software wilt porten en binden.
In principe kunt u volstaan met een eigen driver die afzonderlijke pixels met een gedefinieerde kleur naar het display kan schrijven. *LittlevGL* verwacht een functie die er als volgt uit ziet:

```
/*************************************
* Function    : disp_flush_data
* Description : Sends pixels to the display
* Input       : 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(int32_t x1, int32_t y1, int32_t
   x2, int32_t y2, const lv_color_t *color_array)
{
   /* Here: grab the pixel and send it to the display
   */

   /* Inform the library after job is done */
   lv_flush_ready();
}
```

Deze functie wordt als functie-pointer aan *LittlevGL* overgedragen. De input-parameters zijn de start- en eindcoördinaten van het te vullen tekenbereik, en een pointer naar de beelddata. De daadwerkelijke aansturing van de pixel hangt af van de driver voor het desbetreffende display. Het kan dus zijn

dat de door *LittlevGL* gewenste kleur nog voor het specifieke display moet worden omgerekend, bijvoorbeeld van RGB naar BGR. Meer hoeft de eigenlijke tekenroutine niet te kunnen. En daarnaast zijn er ook oplossingen voor hardware-versnelling, zoals met de DMA2D-engine van sommige STM32-controllers.

Lichtgeraakt

Het moge nu duidelijk zijn hoe een plaatje op het display wordt weergegeven. Wat nog ontbreekt is de verwerking van gegevens van de touch-controller. Daar is een LittlevGL-functie voor die de coördinaten van een eventuele aanraking al naargelang het apparaat uitleest en verwerkt. RPi-displays zijn meestal voorzien van een *XPT2046* touch-controller die als slave met de SPI-bus is verbonden. Deze kan helaas niet worden uitgelezen met een hogere klokfrequentie dan 2,5 MHz. Aangezien het display en de controller op een 20MHz-klok draaien (bij kamertemperatuur tot 26 MHz), moeten we de bus-klokfrequentie aanpassen gedurende de communicatie met de touch-controller, en daarna weer op de oorspronkelijke waarde terugzetten. Ook hier helpt de bibliotheek *TFT_eSPI*, want niet alleen ondersteunt die de XPT2064, maar hij zorgt ook automatisch voor deze klok-omschakeling

Als u geen touch-bediening gebruikt, dan kunt u de UI ook bedienen met de muis, het toetsenbord of met een draai-encoder en druktoetsen. Het spreekt vanzelf dat u voor die invoermethoden ook geschikte drivers nodig heeft. En die moeten op de juiste manier in *LittlevGL* worden geregistreerd.

Opbouw van het beeld

Laten we even stilstaan bij de sterke punten van *LittlevGL*: het is een groot voordeel dat we over de broncode kunnen beschikken, want dat vergemakkelijkt het debuggen van onze eigen code. Bovendien is de bibliotheek goed gedocumenteerd en wordt deze actief verder ontwikkeld. Ook beginners kunnen aan de hand van voorbeelden snel met succes hun eerste resultaten boeken en interfaces maken. Te beginnen met eenvoudige labels via buttons en tabellen tot dropdown-lijsten en schaalverdelingen staat de gebruiker een heel palet aan bedieningselementen ter beschikking. Verder zijn er vensters en tooltips, alsmede thema-sjablonen om de GUI nog beter aan het beoogde doel aan te passen. Alle elementen worden in detail beschreven in de documentatie. Onder [1] worden ook de diverse functies van de library en hun onderlinge samenhang gedemonstreerd. Op het moment van schrijven biedt *LittlevGL* (nog) geen basisfuncties voor het zetten van pixels of het tekenen van lijnen. Dat komt door de manier waarop het beeld wordt opgebouwd: als er een element verandert, wordt het nieuw te tekenen deel van het beeld bepaald en wordt de buffer in het interne RAM voorbereid. Vervolgens wordt het beeld dan naar het display gestuurd. We moeten dus niet alleen een lijn als object ter beschikking hebben, maar individuele pixels. Deze beperking maakt het echter aanzienlijk eenvoudiger om alleen delen van het hele scherm opnieuw te tekenen.

Bekijken we nu de technische details van het genereren en weergeven van beelden. Als we flikker- en storingsvrije animaties of beeldscherm-updates willen, kunnen we het complete beeldscherm in het RAM van de controller voorbereiden en vervolgens deze data naar het display sturen. In ons geval gaat het dan om 307 KB aan data. Men zou echter ook meteen alle elementen naar het beeldscherm kunnen sturen zodat minder RAM in beslag wordt genomen. Dat maakt een flikkervrije weergave echter lastiger en bemoeilijkt effecten als anti-aliasing, transparantie en schaduw.

Tabel 1. ESP32-pinout.

Functie	Display-pin	ESP32-pin	Opmerkingen
MISO	21	19	
MOSI	19	23	
SCK	23	18	
DC	18	02	
CS	24	05	
RST	22	EN	spaart 1 GPIO-pin uit
T_CS	26	04	
VCC (5 V)	02	5 V	
GND	14 / 25	GND	
T_IRQ (optioneel)	11	34	ESP32-pin alleen input

Bij wijze van compromis kunnen we een deel van het beeldschermbereik spiegelen in RAM. Met maar iets meer dan 10% van de geheugenruimte voor het totale beeld hebben we dan toch de beschikking over alle features. Voor een display met 480 x 320 pixels met een 16-bit kleurdiepte is de vereiste geheugenruimte ‘slechts’ 30,7 KB RAM – voor een ESP32 is dat veel maar niet te veel.

In de actuele versie 6 van de libarary wordt het geheugenbereik niet met een `#define` aangegeven, maar moet dit in de code worden gedaan. Deze handelswijze is vooral dan praktisch wanneer er extra extern RAM voorhanden is dat gebruikt kan en moet worden. In ons demoproject beperken we ons tot een statische toewijzing van een bereik in het geheugen van de ESP32, waardoor de code overzichtelijk blijft:

```
//Memory for the displaybuffer
static lv_disp_buf_t disp_buf;
static lv_color_t buf[LV_HOR_RES_MAX * 10];
```

Dit geheugenbereik wordt in de functie hal_init() met de volgende coderegel aan het display toegewezen:

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

Met andere microcontrollers zou u moeten nagaan wat er kan en wat belangrijk is, want een heleboel microcontrollers hebben beduidend minder RAM aan boord of moeten met allerlei kunstgrepen met extern RAM werken. Naast het beschikbare RAM-geheugen zijn ook andere aspecten relevant, zoals de beschikbare rekencapaciteit of mogelijkheden voor multi-threading. *LittlevGL* ondersteunt helaas geen multi-threading, zodat alle operaties via dezelfde thread moeten verlopen als die waarin de functie `lv_task_handler()` wordt aangeroepen. Wat er aan rekencapaciteit nodig is hangt af van de verwachte hoeveelheid interactie en van hoeveel en hoe vaak er op het display getekend moet worden, en of en hoe animaties worden gebruikt. Een ESP32 heeft met zijn twee processor-cores voldoende rekenkracht voor een GUI.

Experimenten

Wie nu zelf wil gaan experimenteren, moet oppassen voor diverse valkuilen. Hieronder volgt daarom een voorbeeld-setup, om die van u zo probleemloos mogelijk te laten verlopen.

Om te beginnen vertoont een ESP32-D4-PICO-board door de extra belasting van het display nog wel eens kuren bij het opstarten. Met een extra condensator van 10 µF tussen 3,3 V en GND vertraagt het bootproces lang genoeg om de spanningen in een gedefinieerd bereik te laten komen.
Het display wordt aan de hand van de pinout in **tabel 1** op het ESP32-board aangesloten. Daarmee is de hardware klaar voor gebruik. Nu volgt de configuratie en test van de software. Als eerste behandelen we de bibliotheek *TFT_eSPI* als eigenlijke displaydriver, en aansluitend gaan we in op de configuratie van *LittlevGL*. Voor de displaydriver zoeken we in de bibliotheek-directory van de Arduino-IDE de map *TFT_eSPI* om het bestand *User_Setup.h* aan het toegepaste display aan te passen. Voor het gebruikte display moeten de volgende `#defines` voorhanden zijn:

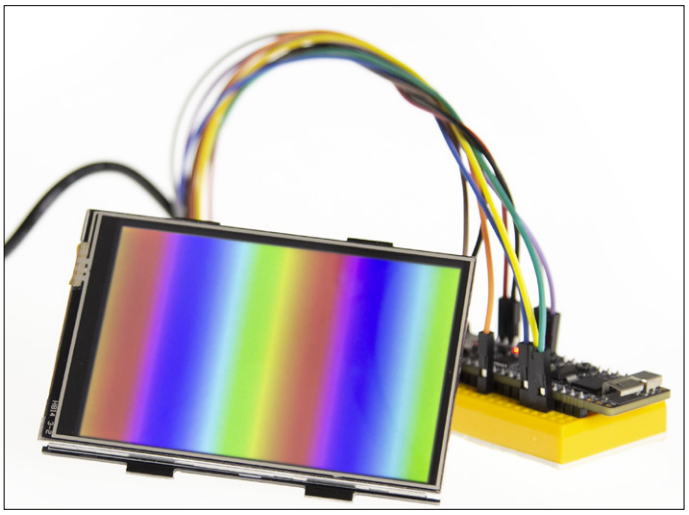
```
#define RPI_ILI9486_DRIVER // max. 20 MHz SPI
#define TFT_MISO 19
#define TFT_MOSI 23
#define TFT_SCLK 18
#define TFT_CS   05 // Chip select control
#define TFT_DC   02 // Data Command control
#define TFT_RST  -1 // set TFT_RST to -1 if display
   RESET is connected to ESP32 RST
#define TOUCH_CS 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
```

Hier definiëren we dus de GPIO's die we gebruiken, en zetten we de SPI-klok op 20 MHz om zeker te zijn dat het display opstart. De klok voor de touch-controller leggen we vast op 2,5 MHz. Voor de test kiezen we een voorbeeld (*TFT_eSPI* → *480x320* → *Rainbow480*) dat de kleuren van de regenboog op het scherm zet. Hebben we een en ander goed aangesloten en is alles gecompileerd, dan moet het scherm er uit zien als in **figuur 2**. Daarmee is dan de hardware in principe klaar voor gebruik.
De volgende stap is dat we *LittlevGL* aan de displaydriver koppelen en onze eigen Human/Machine Interface (HMI) construeren. Om *LittlevGL* te kunnen gebruiken moeten we eerst de configuratie aanpassen. Daarvoor zoeken we in de Arduino-library de map *littlevGL*. Die bevat het bestand *lv_config.h* dat we aanpassen voor ons display en waar de beschikbare bibliotheekelementen worden ingesteld. Aan het begin van dit bestand ziet u de instellingen voor het geheugengebruik. De regel

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

definieert hoeveel RAM voor de GUI-objecten is gereserveerd. De hier aangegeven 64 KB geeft straks problemen met de software-linker, die zal klagen dat er niet zoveel beschikbaar is. Het geheugenbereik van de ESP32 is namelijk geen aaneengesloten blok, en dat merken we bij statisch reserveren (als we gaan compileren). We zouden weliswaar in runtime via `malloc` en `free` de gewenste blokken kunnen reserveren, maar dat brengt andere gevaren met zich mee, en dus pakken we



Figuur 2. Eerste test: het display toont de kleuren van de regenboog.

het anders aan. We veranderen deze regel in:

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

Voor onze eerste stappen is dit genoeg.
Ons display heeft een resolutie van 480 x 320 pixels. De bijbehorende `#defines` zien er zo uit:

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

De resolutie in DPI (dots per inch) volgt uit deze formules.

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

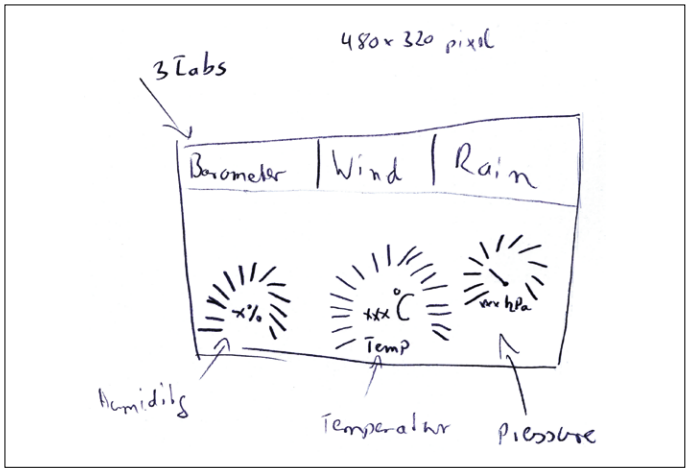
Vullen we alles in, dan levert dat:

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

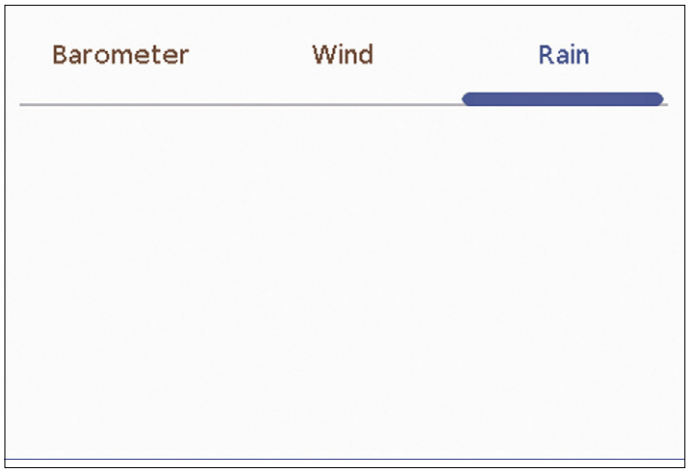
Naar beneden op een geheel getal afgerond:

```
#define LV_DPI 164
```

Tot zover de basisinstelling. Voor de eerste test laten we de overige instellingen ongemoeid. We slaan onze wijzigingen op. In de Arduino-IDE kunnen we nu onder LittlevGL het voorbeeld *ESP32_TFT_eSPI* kiezen en in het ESP32-board laden. Als alles goed geconfigureerd is, verschijnt op het display de tekst "Hallo Arduino!" op een witte achtergrond.
De driver en *LittlevGL* werken dus goed samen. Maar de touch-controller van het display wordt nog niet uitgelezen en de data wordt dus ook nog niet overgedragen naar de bibliotheek. Daarom bekijken we die gedeelten van de code waarmee men het fundament voor een eigen applicatie kan leggen. Laten we het voorbeeld uit de LittlevGL-bibliotheek dat we zojuist in de



Figuur 3. Schets van het display voor de weergave van luchtdruk, temperatuur en luchtvochtigheid, met drie tabs.



Figuur 4. Screenshot met drie nog lege tabs.

ESP32 hebben geladen, *ESP32_TFT_eSPI*, eens nader bekijken. In de functie `setup()` volgt na de initialisatie van de bibliotheek in regel 63 met `lv_init()` en die van het TFT in de regels 69 en 70 met `tft.begin()` en `tft.setRotation(1)` in regel 73 en 74 de initialisatie van de struct `lv_disp_drv_t`. Deze struct krijgt een functie-pointer voor het schrijven naar het display mee en wordt daarna in de bibliotheek geregistreerd.
Iets dergelijks vinden we in de dummy-touch-driver in de regels 80...84. Als laatste stap zetten we met behulp van een 'ticker' een tijdbasis voor de bibliotheek klaar. Hierbij wordt een functie opgeroepen met een vast interval van 20 ms. Telkens wordt een tijdteller met 20 ms opgehoogd. Vervolgens maken we een aanraakvlak ('knop') aan dat de tekst "Hallo Arduino!" toegewezen krijgt in regels 90...92.
Binnen de loop-functie roepen we nu nog `lv_task_handler()` aan, zodat de GUI op invoer kan reageren of het beeldscherm opnieuw kan tekenen.
De auteur heeft een basisproject samengesteld, zodat u niet bij elk project weer vanaf nul hoeft te beginnen. Hierin vinden we de instellingen voor het display van JOY-iT en de bijbehorende touch-controller, en de initialisatie van de diverse onderdelen. In regel 139 van de sketch stellen we de oriëntatie van het display in met `tft.setRotation(3)`. Daarmee wordt het beeld 270° gedraaid ten opzichte van de uitgangspositie. Als een ander display bijvoorbeeld maar 180° gedraaid moet worden, dan moet deze parameter op 1 staan.

GUI

Op dit fundament kunt u nu uw eigen GUI gaan opbouwen. Dat zou u direct op de ESP32-hardware kunnen doen, maar telkens compileren, uploaden en testen kost behoorlijk wat tijd. Het alternatief is een PC-simulator. Daarvoor is wel vereist dat u vertrouwd bent met Eclipse. De installatie is beschreven op [6]. Onder Windows is de installatie iets moeilijker dan onder Linux of OS X. In de simulator kunt u nu uw eerste stappen zetten zonder dat u telkens uw aangepaste code in de hardware hoeft te laden.
Allereerst gaat het om het ontwerp van uw gebruikersinterface. Dat gaat het beste gewoon met potlood en papier (of eventueel met tablet en vinger of stift). Voor de eerste regels code worden ingetypt moeten er schetsen worden gemaakt. In **figuur 3** ziet u zo'n schetsje. Op deze manier is duidelijk waar welke objecten moeten komen en met welke objecten genavigeerd wordt. Het gaat in dit voorbeeld om een weerstation, dus we voorzien een bedieningspaneel met drie tabs voor de weergave van de meetgegevens. Om de Arduino-sketch overzichtelijk te houden zetten we de functies en de opbouw van de GUI-elementen in een eigen bestand.
Als eerste bereiden we het scherm voor voor en maken we een tabview-element aan waarin we vervolgens drie tabs instellen voor luchtdruk (barometer), wind en regen (rain). Voor het voorbereiden van het scherm zorgt deze code:

```
lv_theme_set_current(th);

/* Next step: create a screen */
lv_obj_t * scr = lv_cont_create(NULL, NULL);
lv_scr_load(scr);
```

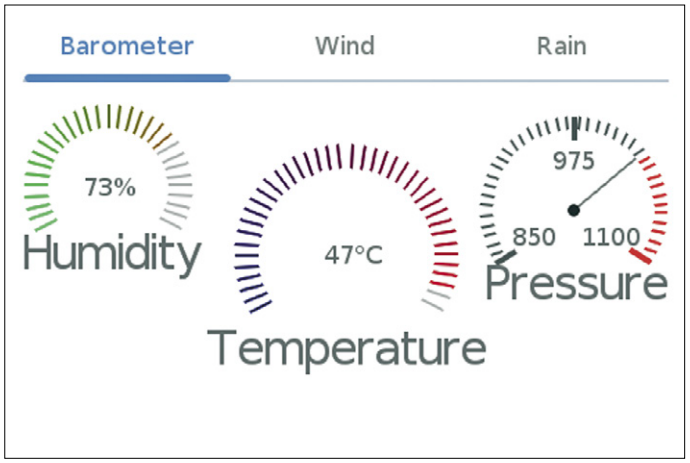
Het sjabloon (theme) wordt geladen, opgegeven als functieparameter. Daarna maken en laden we een leeg scherm. Het tabview-element krijgt het volledige scherm toegewezen. In het screenshot van **figuur 4** ziet u de drie lege tabs, met lettertype en -grootte zoals vastgelegd in het sjabloon. Als we op de naam van een tab klikken, dan met een blauwe markering aangegeven dat die tab nu actief is. Meer dan dat zien we nog niet want de tabs zijn verder nog leeg. Met de volgende regels code:

```
/* And now populate the three tabs */
lv_obj_t * tv = lv_tabview_create(scr, NULL);
lv_obj_set_size(tv, LV_HOR_RES, LV_VER_RES);
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");
```

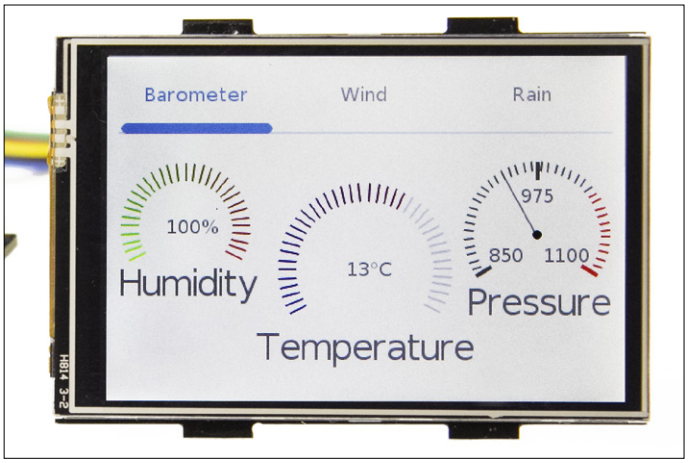
zijn de eerste drie tabs aangemaakt. In eerste instantie hebben ze nog geen inhoud, maar ze hebben al wel titels als parameter meegekregen.

Weerstation

Als eerste is de barometer aan de beurt, die de drie waarden voor luchtvochtigheid, temperatuur en luchtdruk moet tonen.



Figuur 5. Screenshot van de eerste tab met barometerwaarden.



Figuur 6. De barometer in het echt.

Voor luchtvochtigheid en temperatuur gebruiken we `lv_lmeter` en label, waarmee de waarde en de grootheid worden weergegeven. Voor de luchtvochtigheid nemen we `lv_gauge`. Het uiterlijk van de elementen kunnen we in runtime nog aanpassen met verschillende stijlen. Zo kunnen we elk element nog individualiseren.
Bij het rangschikken van de elementen moeten we rekening houden met de autofit-functie van de bibliotheek. Dus ofwel maakt u zelf de elementen passend, ofwel schakelt u autofit uit. U kunt elementen vanuit meerdere begincoördinaten positioneren, op [7] vindt u een overzicht van de diverse mogelijkheden. De afzonderlijke elementen kunnen *parents* hebben, dat is een object waarvan hun positie afhangt. Zo ontstaat een elegante positie-afhankelijkheid die ervoor zorgt dat alle *children* zich opnieuw uitlijnen zodra er een *parent* verschuift. Is een element eenmaal aangemaakt, dan kunt u dat later via de code niet meer rechtstreeks aanspreken. Voor *LMeter* en *Gauges* maken we daarom gebruik van globaal toegankelijke pointers. In de voorbeeldcode

```
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
```

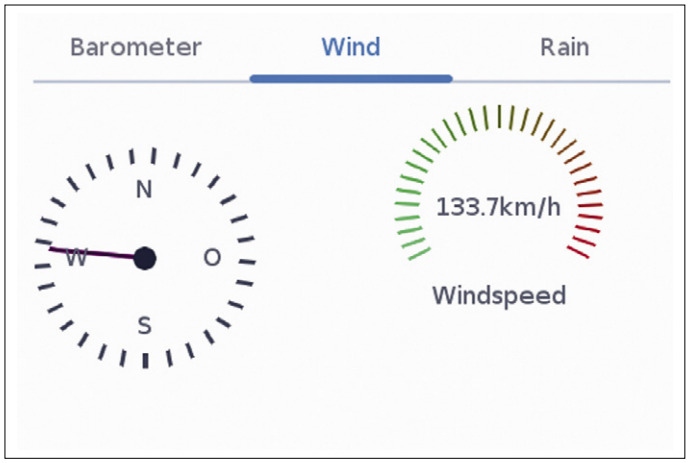
valt op dat functies als `lv_lmeter_create` altijd alleen een pointer teruggeven. De vraag is nu waar geheugen gereserveerd wordt. Dat zit wat dieper verstopt in de bibliotheek. De uitdrukking

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

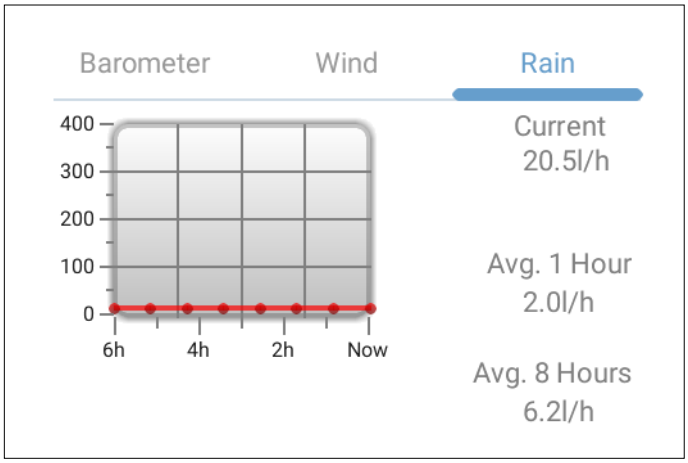
maakt een vaste geheugen-*pool* voor grafische elementen aan. Telkens als er een create-functie wordt aangeroepen, wordt er een stukje geheugen uit de pool voor dat grafische object gereserveerd. Het resultaat is een pointer naar het geheugenadres waarmee de eigenschappen van het object kunnen worden gewijzigd. Als het geheugen op is – wat bij dynamische GUI's kan gebeuren – dan geeft dat een fout in de bibliotheek die naar een eindeloze lus voert. De ESP32 houdt er dan helemaal mee op.
Aan zo'n pointer hebben we alleen iets binnen de betreffende functie. Als we later nog zonder omwegen een element willen kunnen benaderen, dan moeten we die pointers buiten de functie bewaren. Omwille van de eenvoud gebruiken we hier een paar globale variabelen. Voor 'serieuze' toepassingen is het gebruik van globale variabelen echter af te raden.
Via een pointer kunnen we nieuwe meetwaardes naar het display schrijven. Nemen we de functie `UpdateTemperature` als voorbeeld. Het weergave-element *Lmeter* verwacht een waarde tussen 0 en 100, maar onze grootheid (de temperatuur) heeft een bereik van ±50 C°. De temperatuurwaarde moet dus een offset van 50 krijgen, waarbij 0° overeenkomt met een *Lme-*

Weblinks

[1] LittlevGL: https://github.com/littlevgl/lv_arduino
[2] ESP32-weerstation, ElektorLabs januari/februari 2019: www.elektormagazine.nl/180468-01
[3] TFT_eSPI: https://github.com/Bodmer/TFT_eSPI
[4] RA8875-library: https://github.com/adafruit/Adafruit_RA8875
[5] u8g2-library: https://github.com/olikraus/u8g2
[6] PC-simulator: https://docs.littlevgl.com/en/html/get-started/pc-simulator.html
[7] Object-positionering: https://docs.littlevgl.com/en/html/overview/object.html#object-s-working-mechanisms
[8] Gigantische LED-klok, ElektorLabs mei/juni 2019: www.elektormagazine.nl/180254-03
[9] Projectpagina bij dit artikel: www.elektormagazine.nl/190295-04



Figuur 7. Tab met windrichting en -snelheid.



Figuur 8. Screenshot van de tab met het neerslagverloop en -waarden.

ter-waarde van 50. Bovendien geven we de actuele waarde ook nog weer als tekst. Dat doen we met `snprintf` en een kleine lokale buffer die als nieuwe tekst in het tekstveld geschreven wordt. Een langere tekst wordt niet automatisch opnieuw uitgelijnd. Dat moeten we dus zelf doen na elke tekstwijziging, en wel door het aanroepen van de functie `lv_obj_align` met de parameter voor het desbetreffende label. De metertjes voor luchtvochtigheid en luchtdruk gaan op dezelfde manier. In **figuur 5** ziet u een screenshot van de complete tabs met daarnaast in **figuur 6** hoe dat er in het echt uitziet.

Nu hebben we de eerste tab van inhoud voorzien. Bij de tweede tab gaan we net zo te werk, alleen hebben we voor de weergave van de windrichting (met een windroos/kompas) nog wel wat werk te doen. Hier gebruiken we een *gauge* als *parent* voor vier *labels*. In de code definiëren we een *gauge* met een bereik van 0...359°. Daarna volgen vier *labels* waaraan we als parent het kompas geven. We definiëren de *labels* ten opzichte van het midden van het kompas. Dat resulteert in de vier kompasrichtingen; de wijzer wijst in die richting waaruit de wind waait. Bij deze gauge correspondeert 0° echter niet met de waarde 0, maar met de waarde 180. Voor de windsnelheid nemen we weer een *Lmeter* die net als bij de barometer wordt opgezet. U merkt dat de stappen voor het definiëren van elementen steeds dezelfde zijn. Eerst leggen we de stijl van het object vast, dan bouwen we het object op en tenslotte geven we het zijn specifieke eigenschappen. Het eindresultaat ziet u in **figuur 7**.

Bij regen of neerslag willen we de meetwaarden op een andere manier zien. De waarden worden tekstueel en in de vorm van een grafiek als functie van de tijd weergegeven. De teksten worden op dezelfde manier als hierboven gerealiseerd. Eerst definiëren we stijlen en objecten, dan wijzen we de waarden toe. Voor het verloop van de hoeveelheid regen kiezen we een ruitjesdiagram; sinds versie 6 zijn daar geen trucs meer bij nodig om de assen van tekst te voorzien. Voor het actualiseren van de waarden hoeft niet elk datapunt afzonderlijk verplaatst te worden; deze taak neemt `lv_chart_set_next` voor zijn rekening. Er wordt eenmaal per uur een nieuwe waarde naar het diagram gestuurd. De actualisering van de hoeveelheid neerslag gaat net als bij de andere teksten met behulp van een eigen functie. **Figuur 8** toont een screenshot met 'pseudodata' voor het neerslagverloop en de hoeveelheden.

Voor de dataverbinding met het display hebben we code uit het project Gigantische LED-klok [8] hergebruikt die data van een broker via MQTT verwerkt. Deze code verwacht van de broker een JSON-string met daarin de waarden voor luchtvochtigheid, temperatuur, luchtdruk, windrichting, windsnelheid, en de hoeveelheid neerslag. Nieuwe data van de broker worden in de betreffende elementen ingevuld. We moeten er wel op letten dat dit niet in verschillende threads gebeurt. Ook in de configuratie zijn er (afgezien van de ontbrekende klok-instellingen) geen grote verschillen met de LED-klok [8]. De instellingen voor WLAN en MQTT nemen we gewoon over, alleen moeten we het weerstation en het display op hetzelfde topic instellen. Daarna komen de waarden direct op het display. De hoeveelheid regen is op dit moment nog een uitzondering: hier wordt slechtsde actuele hoeveelheid neerslag van het weerstation weergegeven. De berekening van de hoeveelheid per uur en het verloop zijn in dit weerstation helaas nog niet geïmplementeerd. Maar als dat eenmaal gebeurd is, dan zullen ook die waarden op het display verschijnen.

Conclusie

Dit voorbeeld toont enkele basisfuncties van *LittlevGL* in de praktijk. De bijbehorende code kunt u zoals altijd gratis downloaden van de projectpagina bij dit artikel [9]. Zoals al opgemerkt biedt LittlevGL nog veel meer functies en animaties, waaronder weergave met tabellen, lijsten en dropdown-menu's. Spreekt de eenvoud van het gebruik van deze bibliotheek u aan, dan zult u LittlevGL beslist ook met uw eigen projecten willen uitproberen. Een ESP32-module met een touch-display is dan een echt universeel inzetbaar platform dat veel waar voor zijn geld biedt. ⏮

(190295-04)