Displays am Mikrocontroller

von Schnittstellen und Chipsätzen bis zum GUI-Editor

Zahlreiche Mikrocontroller-Projekte haben als Frontend ein Display, von der LED-Matrix bis hin zum E-Ink-Display – klein, rund, groß, oder auch mit eigener Intelligenz an Bord. Wir bieten einen praktischen Überblick über Displayarten, Schnittstellen, Chipsätze, Libraries für die Programmierung und Sonderanwendungen.

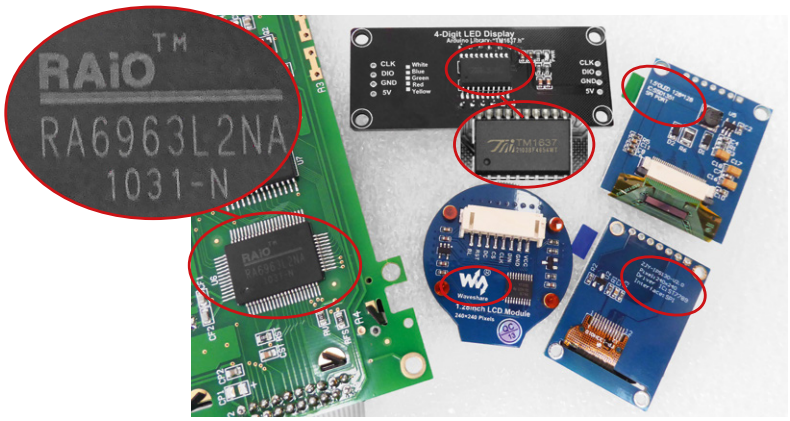

Bild 1: In den meisten Fällen sind die eingesetzten Chipsätze auf den Display-Boards zu finden, ansonsten orientiert man sich am Hersteller

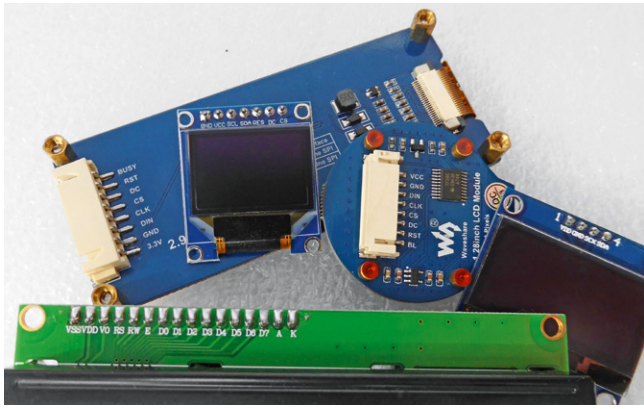

Bild 2: Für den Anschluss von Displays werden unterschiedliche Schnittstellen verwendet, hier 4-/8-Bit-Parallel, SPI, I²C.

Visuelles Interface

Ob Wetterstation, Web-Radio, Sensordatenanzeige, Messenger- und Messwertausgabe – Displays sind heute nicht mehr wegzudenken, wenn es um Mikrocontroller-Applikationen geht. LED-Arrays, LED-Multi-Segment-Anzeigen, LC-Displays – ob fest programmiert, alphanumerisch oder Grafik-/Pixelanzeige, TFT-, OLED- oder E-Ink – die Auswahl ist enorm und gipfelt in den heute verfügbaren, einfach per Editor konfigurierbaren Smart-Displays. Dazu kommt oft die Touch-Option, um auch die Bedienung in das Display zu integrieren.

Die meisten Displays sind heute mit einem spezialisierten Controller ausgestattet, der die Ansteuerung von Pixeln, Farben und der Hintergrundbeleuchtung sowie das Stromversorgungsmanagement übernimmt. So bleibt dem Anwender mindestens eine aufwendige Verdrahtung erspart, aber auch das Pixelmanagement wird ihm abgenommen. Man denke nur an den Anschluss- und Ansteueraufwand von puren Displays mit vieldrähtigen Schnittstellen wie LVDS, RGB parallel usw.

Bei den Controllern gibt es eine große Anzahl verschiedener Type. Letztendlich ist es wichtig zu wissen, welchen Typ man vor sich hat und wie dieser anzusteuern ist. Entweder man erkennt den Chipsatz direkt, oder er ist aufgedruckt, wie in Bild 1 in einigen Beispielen zu sehen ist. Oder man hat Unterlagen bzw. orientiert sich bei Herstellern oder Lieferanten dazu.

Erfreulich übersichtlich: Die Schnittstellen zu den Displaycontrollern. Hier gibt es, abgesehen von Spezialanwendungen, nur einige Standard-Schnittstellen wie die früher als Standard-Schnittstelle bekannte und heute eher wegen der Vielzahl an Anschlussleitungen seltener eingesetzte „Parallel 4/8-Bit"-Display-Schnittstelle, die I²C/TWI-Schnittstelle oder SPI (Bild 2). Typische Video-Schnittstellen wie VGA, Displayport oder HDMI diskutieren wir hier nicht. Einige davon, wie VGA, kann man zwar auch an kleinen Mikrocontrollern wie dem AVR emulieren, aber in der Praxis bleiben sie kompletten Mikrorechnersystemen mit entsprechenden Videoports wie dem Raspberry Pi vorbehalten.

Bleibt noch die Software-Schnittstelle. Sie muss die Ansteuermöglichkeiten des Displays bzw. Chipsatzes wie Adressierung, Zeichensatz, Farbzuordnungen, Helligkeitssteuerung, Grafikbefehle, Datenbus-Handling, interne Zeitabläufe usw. enthalten. Diese Software-Schnittstellen nennt man Libraries, zu Deutsch Bibliotheken (ff.: Library). Bild 3 zeigt einen Ausschnitt aus einer OLED-Library mit der Controllerreihe SSD1306 - SH1106. Sie erleichtern dem Programmierer das Leben enorm, weil sie ihn von lästigen Routinearbeiten deutlich entlasten, sie werden einfach per „include" o. ä. in das Hauptprogramm eingebunden, beanspruchen allerdings auch Rechnerressourcen. Im eigentlichen Anwendungsprogramm muss man dann nur noch die wichtigsten Parameter des jeweiligen Displays wie Pixel, Reset-, Init- und andere Steuerbefehle und die Hardware-Anbindung an den Mikrocontroller angeben.

Dem Display kann man später nur das abverlangen, was in seiner Softwareschnittstelle, der Library, steht. Geschickte Programmierer schreiben hier durchaus anhand des Displaycontroller-Datenblatts eigene Libraries. Deshalb findet man auch über die Standard-Libraries hinausgehende Softwareschnittstellen, etwa mit modifizierten Zeichensätzen. Denn nicht jede Library kann alles, so kann es durchaus vorkommen, dass man etwa ein Grad-Zeichen vergeblich sucht. Ansonsten findet man alle zulässigen Befehle wie Grafikbefehle, Scrollen, Texteingabe usw. in der Library.

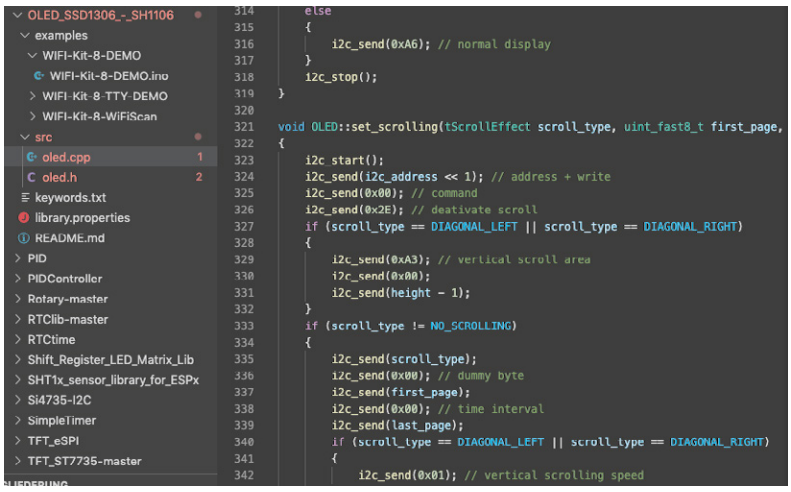

Bild 3: In den Libraries sind alle Informationen, Zuordnungen und Befehle für den jeweiligen Displaycontroller zusammengefasst

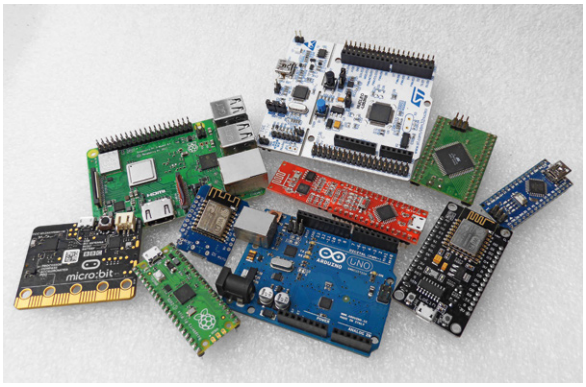

Bild 4: Treiber und Libraries gibt es für alle Mikrocontroller- und SBC-Plattformen. Für die meisten dieser Plattformen gibt es auch direkt mechanisch-elektrisch passende Displays.

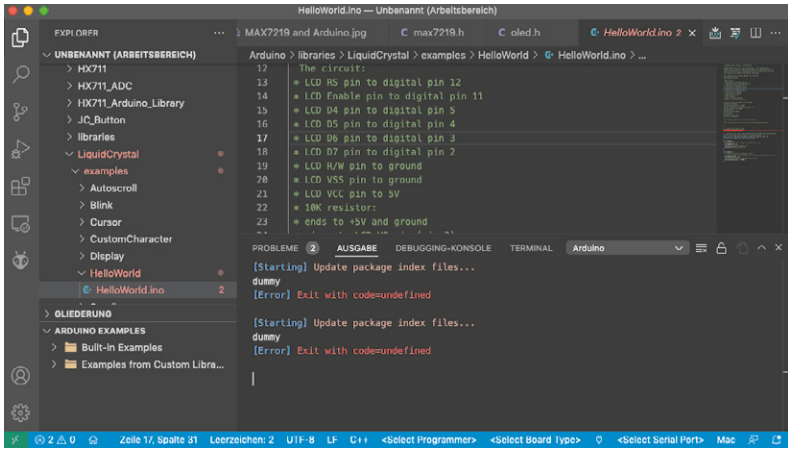

Bild 5: Eine beliebte IDE ist das auf Microsoft Visual Studio Code basierende Platformio

Wir wollen in unserem Exkurs möglichst ballastfrei den Umgang mit Displays in gängigen Mikrocontrollerumgebungen wie AVR/Arduino, Raspberry Pi Pico, micro:bit, ESP, ST32 und an SBC (Single Board Computer = Einplatinenrechner) wie dem Raspberry Pi (Bild 4) vermitteln.

Software/Libraries

Die gängigen Mikrocontroller-Plattformen werden heute in den meisten Fällen direkt per USB bzw. über USB-UART-Adapter und entsprechende IDEs (engl. Integrated Development Environment, dt. Programmier-/Entwicklungsumgebung) programmiert. Von diesen gibt es zahlreiche Varianten und daneben noch Debug-Programme und Editoren. Neben der bekannten Arduino-IDE [1], über die inzwischen auch die beliebten ESP- und RP2040-Plattformen sehr einfach programmierbar sind, und die es auch als Cloud-Editor gibt, gibt es weitere, wie das Java Development Kit (JDK) von Oracle [2], das auf Microsoft Visual Studio Code basierende [3] Platformio (Bild 5), verschiedene C/C++ IDEs, AVR-Entwicklungsumgebungen, bis hin zu einfachen Code-Editoren, z. B. für die Programmiersprachen Scratch [4], Python [5] mit seinen zahlreichen Derivaten, oder dem Microsoft MakeCode Editor [6].

Zunehmender Beliebtheit erfreuen sich heute auch rein grafische Editoren wie z. B. Node Red.

Schließlich soll noch die für ARM-Cortex-Prozessoren wie den STM32 geeignete IDE GNU GCC-ARM mit Eclipse-IDE [7] oder/und Mbed und dem zugehörigen Programmieradapter mit Debugger ST-Link erwähnt werden.

Um das eigentliche Programmieren der Anwenderprogramme kümmen wir uns an dieser Stelle nicht, sondern lediglich um die o. a. Libraries.

Die meisten Controller-Entwicklungsplattformen nutzen die für die Arduino-IDE von verschiedenen Entwicklern bereitgestellten Libraries. Eine gute Startseite für die Suche nach einer Library ist immer die der Arduino Foundation, auf der man alle Standard-Libraries und Verweise auf Weitere findet. Viele davon sind bereits im Installations- bzw. Update-Umfang der Arduino-IDE vorhanden. Andere findet man in den Github-Repositories z. B. von Sparkfun, Waveshare (bzw. überhaupt bei den Displayherstellern) oder besonders viele von Adafruit bzw. an Adafruit liefernden Entwicklern. Am Schluss dieses Beitrags findet sich dazu eine kompakte Zusammenstellung.

Das Laden von Libraries in der Arduino-IDE ist über den Library-Manager (Bibliotheken-Verwaltung) sehr einfach. Libraries, die nicht über diesen gelistet sind, lassen sich auch manuell im entsprechenden Arduino-Libraries-Ordner auf dem PC installieren. Hinweise dazu finden sich ebenfalls bei der Arduino-Foundation.

Das Laden der Library in das Anwendungsprogramm erfolgt über den #include-Befehl, z. B. mit:

```
#include <Adafruit_SSD1306.h>
```

Dem folgen dann im Header über den #define-Befehl zusätzliche Spezifizierungen, um das richtige Display zu definieren. Denn in den Libraries befinden sich meist alle mit dem Displaycontroller, hier z. B. der OLED-Controller SSD1306, bestückten Displays mit verschiedenen Auflösungen usw. So muss man dann zum Beisiel für ein 64 x 32-Display die folgende Zeile gültig stellen, alle anderen werden mit // auskommentiert:

```
#define SSD1306_64_32
//define SSD1306_128_64
```

Für Python gibt es, z. B. bei Adafruit oder etwa Random Nerd Tutorials speziell angepasste Libraries, hier auch Driver genannt, die über den Python-Editor in den Controller geladen und im dort abgelegten Python-Programm eingebunden werden.

Beim Raspberry Pi, der ja auch Python Libraries verwendet, erfolgt das Laden der Library im Terminalprogramm von der entsprechenden Quelle z. B. so:

```
git clone https://github.com/adafruit/Adafruit_Python_SSD1306.git
```

Danach wird sie installiert:

```
cd Adafruit_Python_SSD1306
sudo python setup.py install
```

und kann dann in ein Python-Programm integriert werden.

Besonders einfach erfolgt das Einbinden einer Bibliothek in die blockbasierten Einsteigerprogramme wie z. B. in Bild 6 mit dem BBC micro:bit und dem speziell dafür entwickelten Waveshare-TFT-LCD-Board mit 1,8"-Diagonale und einer Auflösung von 160 x 128 Pixeln zu sehen. Über die „ad Package"-Funktion von MakeCode wird das via Waveshare-Github bereitgestellte Package „WSLCD1in8" geladen und steht sofort zur Nutzung zur Verfügung. Der Screenshot in Bild 6 zeigt ein dann blitzschnell zusammengestelltes Programm dazu.

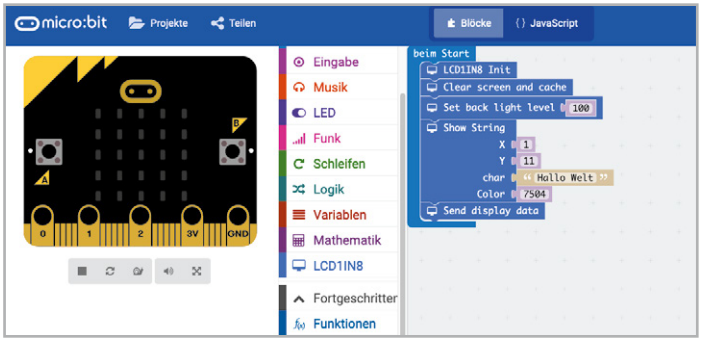

Bild 6: Komplette Lösung – #einfach zu programmieren: Das Display für den BBC micro:bit. Bild Display: Waveshare

LC-Displays

Technik, Aufbau und Schnittstellen

Trotzdem sie zunehmend von Aktiv-Matrix-TFT- und OLED-Displays verdrängt werden, spielen passive LC-Displays in ihrer Form als zeilen- und spaltenorientierte Punktmatrix-Displays immer noch eine wesentliche Rolle. Der verbaute Displaycontroller ist essentiell für die Ansteuerung des Displays. Herkömmliche (Punktmatrix-)LC-Displays, die zeilen- und spaltenweise in einer oder mehreren Matrix-Zeilen organisiert sind, arbeiten meist mit dem Standard-Controller HD44780 bzw. seinen Äquivalenten. Ihm zur Seite stehen, wie in Bild 7 zu sehen, bei größeren Displays Treiberbausteine, meist der HD44100. Der HD44780 kann Punktmatrix-Displays mit bis zu 16 Zeilen und 40 Spalten treiben, die sich auf zwei Zeilen mit einer Punktmatrix (auch Dotmatrix) von 5 x 8 Pixeln aufteilen. Soll das Display größer sein, werden o. a. Treiber und ggf. mehrere Controller eingesetzt, so kommen dann etwa die verbreiteten Größen wie 128 x 64 Pixel zustande.

Auch der darstellbare Zeichenvorrat ist in einem ROM-Bereich des Controllers hinterlegt, er ist im jeweiligen Datenblatt des Displays (nicht des Controllers!) hinterlegt und kann je nach Typ und Hersteller des Displays unterschiedlich ausfallen. Jedoch ist der Anwender nicht allein auf diesen Zeichenvorrat beschränkt, denn es gibt auch einen frei mit Zeichenmustern beschreibbaren RAM-Bereich. Dies ist nützlich, wenn man etwa ein Display hat, in dessen ROM keine Umlaute hinterlegt sind, oder wenn man spezielle Sonderzeichen benötigt.

Der Controller wird über eine weitgehend genormte parallele Schnittstelle (Tabelle 1) mit 14 Pins angesteuert, dazu kommen bei hinterleuchteten Displays zwei Pins für die Beleuchtung. Weitgehend, weil es herstellerspezifische Anschlussreihenfolgen und auch, z. B. bei Einsatz eines zweiten Controllers, zusätzliche Enable-Leitungen für die Controller gibt. Meist ist die Anschlussbelegung auf die Trägerplatine aufgedruckt, ansonsten muss man das Datenblatt zu Rate ziehen.

Zum bereits erwähnten Stichwort 4-/8-Bit-Display noch einige Worte. Hier ist allein die Ansteuerungsart des Displays gemeint. Man kann das Display komplett parallel über die Nutzung aller acht Datenbits und entsprechenden Ports DB0 bis DB7 ansteuern. Dies hat jedoch mitunter Nachteile. Man muss zur kompletten Ansteuerung acht Datenleitungen, dazu RS (Reset), R/W (Read/Write) und Enable bereitstellen. Das sind 11 Adern plus Stromversorgung und evtl. Hintergrundbeleuchtung – ein regelrechter Kabelbaum. Zweiter Nachteil: Acht Datenleitungen belegen acht Ports des Mikrocontrollers, und die stehen oft nicht zur Verfügung. Deshalb gibt es den 4-Bit-Modus, bei dem nur DB4 bis DB7 genutzt werden und die zu schreibende Information auf 2 x 4 Bit (1x obere, 1x untere 4 Bit) mit einer entsprechenden Flusssteuerung aufgeteilt wird. Die Abläufe und die gesamte Displaysteuerung sind in der zugehörigen Library hinterlegt, man muss im Anwendungsprogramm nur den gewünschten Mode einstellen, um das Display entsprechend zu initialisieren und später anzusteuern.

Eine Besonderheit ist die Spannungsversorgung bzw. die Kontrasteinstellung. 5 V ist (mit wenigen Ausnahmen) der Standard für die Spannungsversorgung. Dazu erfordern normale Displays eine positive Kontrastspannung im Bereich bis 1,5 V am Kontrastanschluss Pin 3.

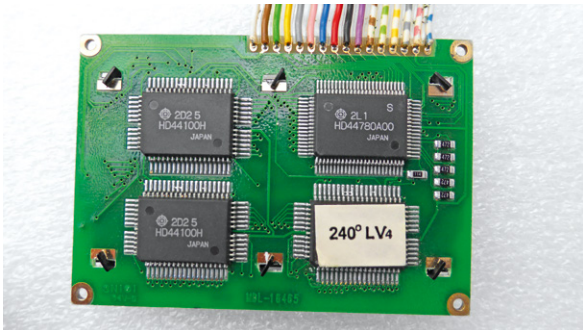

Bild 7: Mehrzeiliges LC-Display mit HD44780-Controller und zusätzlichen Zeilentreibern

Komplizierter wird es bei den sogenannten Hochtemperaturdisplays (der Kontrast und Blickwinkel bei LC-Displays ist stark temperaturabhängig, HT-Displays weisen gegenüber normalen Displays einen erweiterten Temperaturbereich nach oben und unten auf, z. B. -20 °C bis +70 °C) – sie erfordern eine negative Kontrastspannung zwischen -2 V und -5 V. Ist die nicht in Form einer integrierten Ladungspumpe auf dem Displayboard realisiert, muss man eine externe Negativ-Spannungserzeugung realisieren. Diese Displays sind mit einem gesonderten Aufkleber wie dem in Bild 7 gezeigten Displayaufkleber „240°LV4" gekennzeichnet.

Wenn das Display über eine Hintergrundbeleuchtung (Backlight) verfügt, ist diese über einen Vorwiderstand, z. B. 220 Ω (genauer Wert, und ob dieser Vorwiderstand nicht schon auf dem Board vorhanden ist, siehe jeweiliges Datenblatt oder Aufdruck) an +5 V anzuschließen.

Tabelle 1

Anschlussbelegung Display-Parallelschnittstelle

Pin	Belegung
1	Vss (GND)
2	VDD (+5 V)
3	V0 (Kontrast)
4	RS
5	R/W
6	Enable
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7
15	LED Anode (+5 V, ggf. über Vorwiderstand)
16	LED Anode (GND)

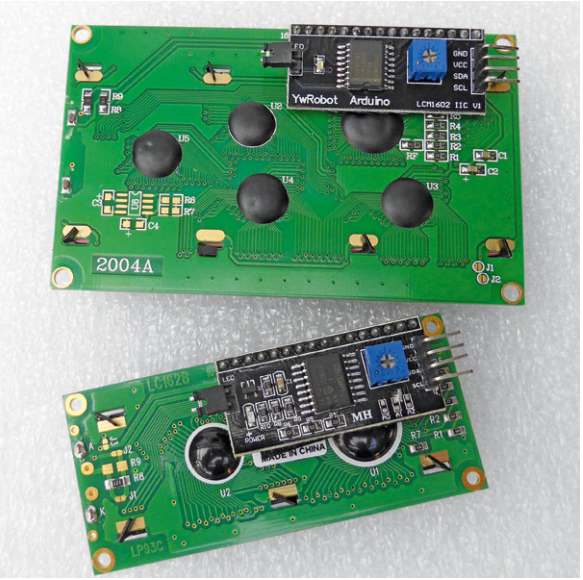

Bild 8: Macht aus 8-Bit-Parallel I²C - der I²C-Portexpander PCF8574

Schließlich noch ein Wort zu weiteren LCD-Controllern. Der HD44780 ist weitgehend Standard, es gibt jedoch eine Reihe weiterer kompatibler Controller, die vor allem von Samsung und Toshiba entwickelt wurden. So trifft man den KS0073, den KS066U oder den leistungsstarken 64-Bit-Low-Power-Chipsatz KS0108B/KS0107 B, der, wie auch der ST7036U, in hoch aufgelösten DOG- und DIP-Modulen, die als System einen sehr geringen Strombedarf aufweisen, eingesetzt wird. Letztgenannter Controller verfügt zusätzlich u. a. über eine I²C-Schnittstelle.

Bei größeren LC-Displays, z. B. 240 x 128 oder größer findet man den ebenfalls mit einem Parallel-Interface arbeitenden T6963 von Toshiba, der auch durch eine umfangreiche Speicherausstattung und die Möglichkeit das Anschlusses von externem sRAM hervorsticht.

Was aber kann man machen, wenn man ein Display mit 8-Bit-Parallelschnittstelle einsetzen möchte, aber nicht mehr entsprechend viele Ports am Mikrocontroller frei hat? Die 4-Bit-Lösung haben wir schon diskutiert, noch einfacher und mit weniger Verdrahtungsaufwand geht es für vielfach eingesetzten LC-Displays der Typen 1602 (2 Zeilen/je 16 Stellen) und 2004 (4 Zeilen/je 20 Stellen) mit einem I²C-Portexpander wie dem PCF8574. Er setzt den I²C-Bus auf die 8-Bit-Parallelschnittstelle um.

Für das direkte Aufsetzen auf ein Display mit einreihiger Anschlussleiste gibt es fertig aufgebaute PCF8574-Boards (Bild 8), die meist auch die Möglichkeit enthalten, die Adressierung des Bausteins am I²C-Bus zu ändern sowie direkt die Kontrasteinstellung vornehmen und die Hintergrundbeleuchtung an- und abschalten zu können. Auch findet man heute LC-Displays gleich ab Hersteller mit dieser Lösung im Huckepack.

Schließlich noch ein Hinweis zu den Logikpegeln in einzelnen Systemen: Hier ist immer darauf zu achten, dass Logikpegel von Mikrocontroller/SBC zu denen des Displays passen, ansonsten muss man einen sogenannten Level-Shifter (Bild 9) einsetzen.

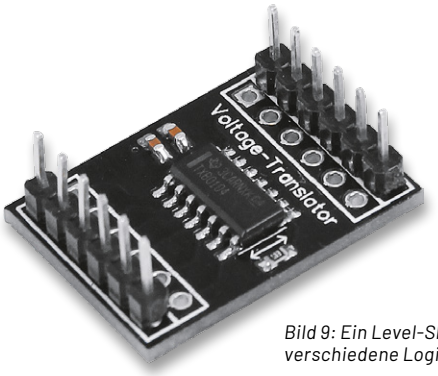

Bild 9: Ein Level-Shifter setzt verschiedene Logikpegel um.

Software/Libraries

Für das direkte Ansteuern eines LC-Displays über die 4- oder 8-Bit-Parallelschnittstelle wird in der Arduino-IDE die Library „LiquidCrystal" eingesetzt und eingebunden:

```
#include <LiquidCrystal.h>
```

Dem folgt die Pinbelegung des Mikrocontrollers, hier für 4-Bit. Dabei ist das Display wie folgt angeschlossen:

- LCD RS an Pin 12
- LCD Enable an Pin 11
- LCD D4 an Pin 5
- LCD D5 an Pin 4
- LCD D6 an Pin 3
- LCD D7 an Pin 2
- LCD R/W und VSS an GND
- LCD VCC an +5V

Nicht vergessen: 10-kΩ-Trimmpoti zwischen +5 V und GND, Schleifer an Display PIN 3 (Kontrast) und ggf. LED-Backlight anschließen, sonst sieht man zunächst nichts im Display.

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

und das Setup für das Display, hier 1602, also 16 Zeichen, 2 Zeilen:

```
void setup()
{
lcd.begin(16, 2);
}
```

Will man das LC-Display wie beschrieben via I²C und PCF8574 ansteuern, nutzt man die Library „LiquidCrystal I2C", die man über den Library-Manager lädt. Hier muss man dann zusätzlich die Arduino „Wire"-Library laden, diese ist bereits in der IDE-Software enthalten. So sieht dann die Einbindung und Initialisierung aus:

Zuerst „Wire" und „LiquidCrystal_I2C" einbinden:

```
#include <wire>
#include <LiquidCrystal_I2C.h>
```

Dann wieder den Displaytyp und die I²C-Adresse des PCF8574 angeben:

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

Nun die Initialisierung und ggf. Einschalten der Hintergrundbeleuchtung:

```
void setup()
{
lcd.init();
lcd.backlight();
}
```

Falls die I²C-Adresse nicht bekannt ist, ermittelt man diese bei angeschlossenem Display mit einem I²C-Scanner, z. B. dem unter [8] verfügbaren Software-Tool.

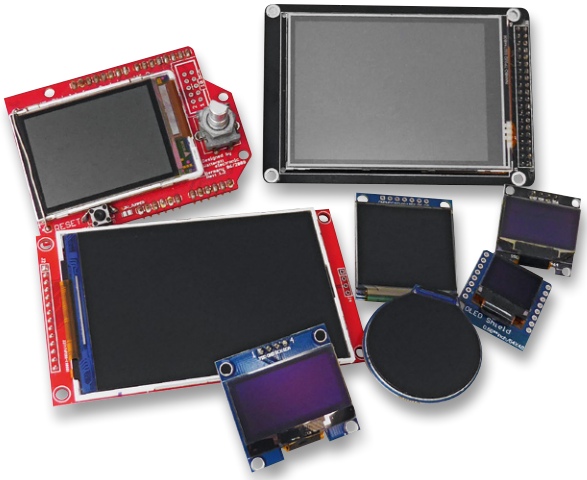

Bild 10: TFT- und OLED-Displays in verschiednen Ausführungen, davon einige mit integriertem Touch-Screen

TFT- und OLED-Displays

Technik und Controller

Die gegenüber traditionellen LCD wesentlich schlankeren TFT- (Aktivmatrix-LCD) und OLED-Displays spielen heute die Hauptrolle in Mikrocontroller-Anwendungen. Es gibt sie in monochrom, manchmal per Software invertierbar, in zwei Farben, z. B. Gelb-Grün (im Grunde genommen, auch monochrom, da meist nur bestimmte Bereiche in der jeweiligen Farbe anzeigen), oder vollfarbig. Auch die Größen und Formen sind sehr vielfältig: von 0,96" bis 10" und größer, eckig oder rund, in Größe und Form an eine bestimmte Hardwareplattform angepasst und/oder mit Touch-Screen. Bild 10 zeigt eine kleine Auswahl dieser Displays in unterschiedlichen Ausführungen.

Auch diese Displays haben jeweils eigene Controller, die sich auf den Trägerplatinen befinden. Anhand dieser Controller wählt man später eine entsprechende Library zur Ansteuerung.

Die weitaus meisten dieser Displays verfügen über die nur wenige Leitungen benötigenden I²C- oder SPI-Schnittstelle. Hintergrundbeleuchtungen und interne Hilfsspannungserzeugungen befinden sich mit auf der Displayplatine, die damit sehr problemlos einsetzbar ist. So muss man im Grund nur wenige Eckdaten kennen: Typ, Auflösung, Controller und Anschluss.

Als Displaycontroller kommt ein ganze Anzahl an Typen zum Einsatz, der Typ ist abhängig von der Displayart (OLED/TFT), der Auflösung und der/den Ausgaben, also monochrom oder farbig. So findet man im OLED-Bereich z. B. den SSD1306, den SSD1331 oder den SSD1351, im TFT-Bereich ist dies z. B. der ILI9488, der ILI9341, der ILI9325, der ST7735 oder ST7736S.

Bei den TFT-Displays taucht vermehrt der Zusatz „IPS" auf. Gegenüber der herkömmlichen TN-Technik von TFT-Displays – hier werden die Flüssigkristalle bei Anregung senkrecht zur Anzeigefläche gedreht – weisen IPS-Displays (IPS = In plane switching) einen größeren Betrachtungswinkel und eine brillantere kontsratreichere Farbdarstellung auf. Die Flüssigkristalle werden hier parallel zur Anzeigefläche ausgerichtet. Vergleicht man einen normalen TFT-Bildschirm in TN-Technik und einen IPS-Bildschirm aus einem seitlichen Betrachtungswinkel, sieht man beim IPS-Display die volle Farbpalette und ein kontrastreiches Bild. Beim TN-Display fallen dagegen hier Farbverschiebungen auf.

Die Schnittstellen

Wie bereits erwähnt kommen nahezu ausschließlich die Schnittstellen I²C und SPI zum Einsatz. Bild 11 zeigt einige Varianten, einmal ein Display mit I²C-Anschluss an einem ESP32 und einmal ein Display mit SPI-Anschluss an einem Arduino und einmal am Raspberry Pi.

Die Displays mit SPI-Schnittstelle erfordern einige Leitungen (DC, CS, RES) mehr, auf der anderen Seite ist die SPI-Schnittstelle schneller, was z. B. bei der Ausgabe von komplexen Grafiken, bei Bildern/Videos oder beim Wechsel von Anzeigen in Touchscreen-Anordnungen entscheidend sein kann. Durch die Chip-Select-Ansteuerung (CS) entfällt bei SPI die bei I²C übliche Adressierung am Bus (siehe Kapitel „LC-Displays"). Auch hier ist die Anpassung von Logikpegeln, wie im Kapitel „ LC-Displays" beschrieben, zu beachten.

Software/Libraries

Für das Ansteuern von OLED- und TFT-Displays werden unterschiedliche Libraries benutzt. In die Arduino-IDE muss man die dem Display-Controller entsprechende Library einsetzen und die Art der Schnittstelle beachten. Für I²C setzt man, wie bereits beschrieben, die Wire-Library „Wire.h" ein, für SPI die SPI-Library „SPI.h".

Betrachten wir zunächst einmal die Einbindung und Initialisierung eines via SPI angeschlossenen TFT-Displays:

Die TFT-Library befindet sich bereits im Umfang der Arduino-IDE:

```
#include <SPI.h>
#include <TFT.h>
```

Je nach eingesetztem Controller-Board ordnet man nun die Anschlüsse CD, DC und RESET zu, entweder standardmäßig über hardwaremäßig vorgesehene Pins der Controller-Plattform oder softwaremäßig zugeordnet. Das sieht dann z. B. beim Arduino Uno so aus:

```
#define CS   10
#define DC   9
#define RESET  8

TFT myScreen = TFT(CS, DC, RESET);
```

Weitere detaillierte Hinweise, z. B. auch für die Nutzung des auf manchen Display-Shields befindlichen SD-Kartenslots sowie Programmbeispiele finden sich unter [9].

Das nächste Beispiel behandelt das Thema OLED-Display mit dem weit verbreiteten Controller SSD1306 und I²C-Anschluss. Um die grafischen und Text-Möglichkeiten des Displays voll zu nutzen, muss hier zur eigentlichen Library immer noch die Library „Adafruit_GFX.h" geladen werden.

Zunächst also die benötigten Libraries laden:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Dann ist mitzuteilen, um welche Displaygröße es sich handelt, hierfür ein 128x64-Display:

```
#define SSD1306_128_64
```

Schließlich braucht das Programm die I²C-Adresse:

```
#define I2C_ADDRESS 0x3C
```

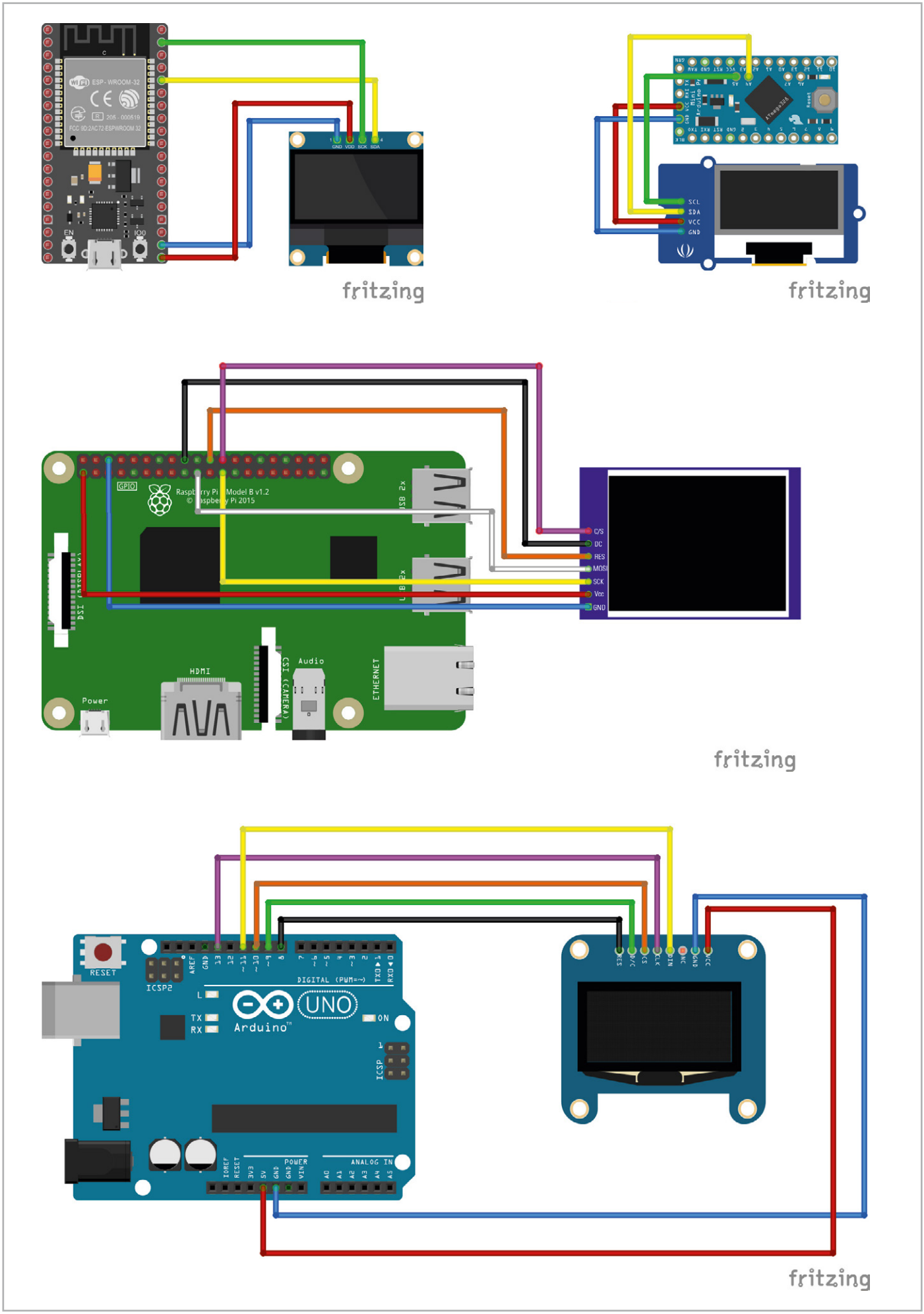

Bild 11: Beispiele für den Anschluss verschiedener Mikrocontroller an Displays mit unterschiedlichen Schnittstellen. Grafik erstellt mit fritzing.

In gleicher Weise entwirft man die Initialisierung für einen ESP32, der über die Arduino IDE programmiert werden soll. Hier nutzt man D22 für SCL und D21 für SDA.

Will man eine Library über den Library-Manager laden, bieten sich hier sehr viele passende Libraries an. Zu deren Nutzung sollte nicht unerwähnt bleiben, dass viele Libraries, z. B. die sehr universell einsetzbare „SSD1306.h" und besonders die sehr viele Displaycontroller einbindende „U8g2", auch viel Speicherplatz auf dem Anwendungssystem benötigen. Da das meist ja eine ganz bestimmte Aufgabe zu erfüllen hat und nur mit gerade diesem einen angeschlossenen Display arbeiten soll, sollte man entweder die Library selbst „abspecken", also alle nicht benötigten Teile entfernen, oder aber spezielle Libraries benutzen. Wenn man z. B. nur ein Textdisplay betreiben will, kann man z. B. „SSD1306Ascii" benutzen.

Einen guten Einstieg in die OLED-Programmierung bietet u. a. [10] mit vielen Anwendungs- und Testprogrammen.

Die Programmierung für den Anschluss von Displays an den Raspberry Pi ist deutlich umfangreicher, weshalb wir hier lediglich auf ein komplettes Tutorial verweisen [11].

Werfen wir abschließend noch einen Blick auf den Betrieb eines I²C-OLED-Displays mit dem SSD1306 und 128 x 64 Pixeln am kleinen Raspberry Pi Pico. Als Schnittstelle dienen bei diesem die Pins 1 und 2 (GP 0 = SDA/GP1 = SCL) der I²C-Schnittstelle I2C0. Für den Betrieb des Displays benötigt man hier einen Mikropython-Treiber, den ssd1306.py, den man neben weiteren Displaytreibern unter [12] findet. Diesen kopiert man sich in das Micropython-Verzeichnis.

Dann erfolgt die Initialisierung mit:

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

i2c = I2C(0, sda=Pin(0), scl=Pin(1), freq=40000)
oled = SSD1306_I2C(128,64,i2c)
```

E-Ink-Displays

Technik und Controller

E-Ink- oder ePaper-Displays sind passive, reflektive Displays, die lediglich das Umgebungslicht zum Ablesen benutzen. Basis ist eine (oft flexible) Polymerschicht, die aus unzähligen Polymer-Mikrokapseln besteht, die unterschiedlich geladene Partikel enthalten (weiß = positiv; schwarz/farbig = negativ). Durch Anlegen eines elektrischen Feldes wandern je nach Ansteuerung die weißen oder die schwarzen Partikel an die Oberfläche. Ergebnis ist eine sehr scharfe und kontrastreiche Darstellung, die wie auf Papier gedruckt wirkt und auch aus einem weiten Betrachtungswinkel ablesbar ist. Bild 12 zeigt ein solches Display von einem der größten Hersteller „Waveshare".

Einmal angeordnet, bleiben die Pixel auch ohne Stromzufuhr über lange Zeit stabil stehen. Das in Bild 12 abgebildete Display wurde z. B.

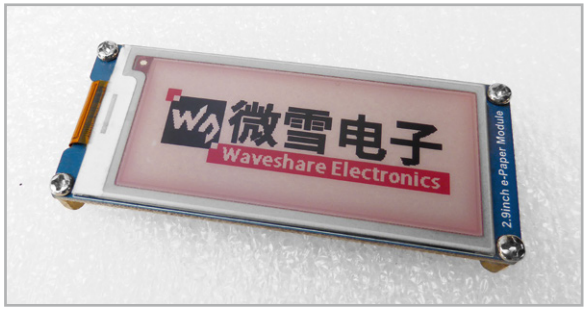

Bild 12: Ein ePaper-Display mit drei „Farben" und noch relativ geringer Auflösung. Das Display hat schon eine Lagerzeit von zwei Jahren hinter sich und zeigt immer noch das werkseitig programmierte Bild an.

vor zwei Jahren gekauft und testweise zunächst nicht genutzt. Die Darstellung hat sich, bis auf eine leichte Rötlichfärbung, nicht verändert. Auch deshalb erfreuen sich diese Displays zunehmender Beliebtheit – sie benötigen, solange man die Darstellung nicht verändert, keinen Strom. Sie sind in der heutigen Form auch nur für relativ langsame Inhaltswechsel einsetzbar, aber eben sehr nützlich für weitgehend statische Anzeigen.

Auch diese Displays arbeiten mit einem internen Controller, dessen Typ aber für uns als Nutzer, der sein Anwendungsprogramm erstellt, zunächst keine Rolle spielt.

Die Schnittstelle und die Libraries

Die Displays verfügen über eine SPI-Schnittstelle, die wie im Kapitel „TFT/OLED" beschrieben, anzuschließen ist. Oder man benutzt eine spezielle Interface-Platine (HAT), die es für den Raspberry Pi und den Arduino gibt (Bild 13). Denn hier ist zu beachten, dass manche dieser Displays mit 3,3 V zu betreiben sind. So sind Level-Shifter und eine 3,3-V-Spannungsversorgung notwendig. Im Wiki auf [13] sind dazu für jede gängige Controller-/SBC-Plattform genaue Anweisungen hinterlegt. Von dort aus findet man auch zu den zugehörigen Libraries und Programmbeispielen. Im Github dazu [14] sind auch die Anschlussbelegungen unter „ConnectingHardware.md" für zahlreiche Plattformen und Mikrocontroller aufgeführt, ebenso Hinweise zur Einbeziehung der Adafriut-GFX-Library für die Behandlung von Schriftfonts und Grafiken.

Wichtig ist, die Auflösung des Displays zu kennen, da es für jedes Display eigene Libraries gibt.

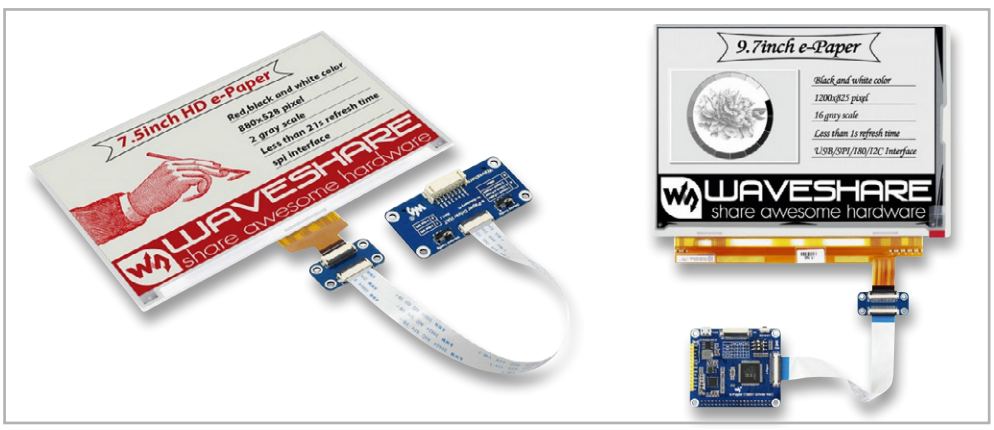

Bild 13: Passt direkt – HATS für den Anschluss eines ePaper-Displays an den Raspberry Pi und den Arduino. Bilder: Waveshare

LED-Displays

Technik, Controller, Schnittstellen

LED spielen neben den bisher beschriebenen Displaytechniken nach wie vor eine Rolle in der Elektronik, man denke dabei nur an LED-Matrixanordnungen. Diese finden wir in unterschiedlichen Formen. Da gibt es einmal die „intelligenten" RGB-LEDs für größere Vollfarbanzeigen, Stripes usw. wie z. B. die WS2812. Sie basieren auf einem eigenem Controller mit PWM-Register, der entweder direkt in die LED integriert ist (WS2812) oder der eine extern angeschlossene RGB-LED ansteuert (WS2811). Weiter gibt es auch RGBW-LEDs dieser Bauart, z. B. SK6812. Von Einzel-LEDs abgesehen, sind diese LEDs immer seriell verschaltet. Die Ansteuerung erfolgt asynchron seriell über nur eine Datenleitung mit insgesamt 24 Bits je LED mit einer Folge unterschiedlich langer High-Impulse. Diese werden zuerst in die erste LED geschrieben, dann ein Low-Reset-Impuls gesendet, worauf hin die gesendeten Bits in das PWM-Register des LED-Controllers übernommen und die LEDs angesteuert werden. Der nächste Taktzyklus wird dann an den nächsten LED-Controller weitergegeben, usw., bis die im Programm einzugebende Anzahl der LEDs der Anordnung erreicht ist. Dann beginnt der nächste Zyklus. Bild 14 zeigt eine mit diesen LEDs aufgebaute 8x8-Matrix , einen Teil eines LED-Stripes mit WS2812 und einen WS8211-Controller.

Eine gewisse Ähnlichkeit hat das Übertragungsverfahren bei (monochromen) LED-Anordnungen mit Schieberegistern (Bild 15). Hier wird ein Bitmuster seriell in das Schieberegister übertragen und mit einem Übernahmeimpuls in die LED-Matrix übernommen.

Diese LED-Anordnungen in der Formation 8x8 können aber auch über den LED-Treiber MAX7219 (Bild 16) via SPI angesteuert werden. Dieser Schaltkreis ist universell für LED-Anordnungen mit maximal 64 LEDs einsetzbar, also findet man ihn auch als Interface in Siebensegment-Anzeigen (Bild 17).

Das SPI-Interface ist hier meist in stark vereinfachter Form vorhanden, es gibt keine MISO-Leitung und selbst die Chip-Select-Leitung CS wird vielfach fest auf dem LED-Modul verdrahtet, z. B., wenn es sich um ein Stand-alone-Modul handelt, das nicht kaskadiert wird.

Die LED-Matrix-Anzeigen werden oft aber auch kaskadiert, weshalb wir hier (Bild 15) die CS-Leitung sehen, die, wie schon weiter vorn besprochen, am Arduino Uno standardmäßig an Pin Digital10 oder aber an einen Pin, der im Programm frei bestimmt wird, zu legen ist. Insgesamt können wir also auch eine solche Anordnung mit gerade zwei bzw. drei Leitungen ansteuern.

Schließlich wollen wir noch einen beliebten Treiber-Schaltkreis betrachten, nämlich den TM1638. Er wird ebenfalls über eine serielle Schnittstelle angeschlossen, die mit dem SPI-Protokoll angesteuert wird. Der Treiber ist in der Lage, bis zu acht Siebensegment-Anzeigen sowie bis zu Einzel-LED anzusteuern, und bis zu acht Taster abzufragen. Hierzu gibt es entsprechend voll ausgerüstete Shields, aber auch die in Bild 18 gezeigte einfache und über Chip Select kaskadierbare Anordnung mit acht Siebensegment-Anzeigen ist weit verbreitet. Bezüglich des Anschlusses der CS-Leitung gilt auch hier das bereits Gesagte.

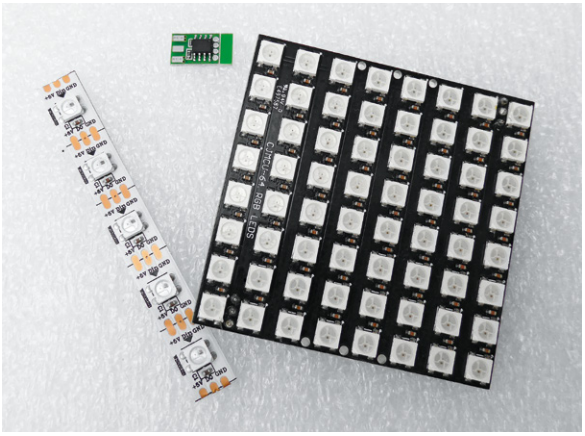

Bild 14: Die mit integriertem Controller ausgestatteten RGB-LEDs werden seriell über nur eine Leitung angesteuert.

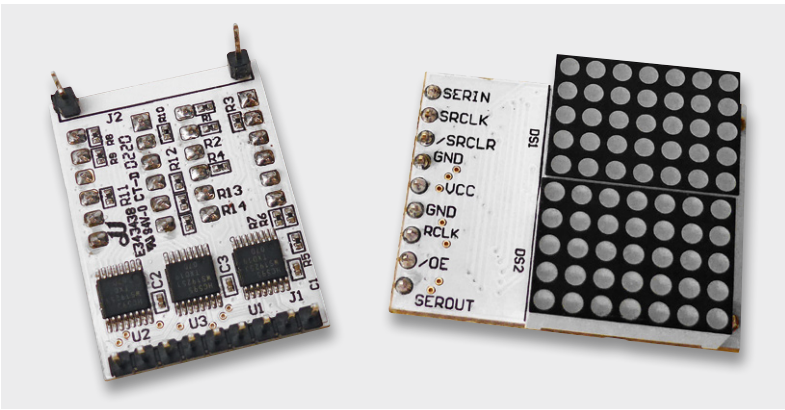

Bild 15: Ein Beispiel für den Einsatz von Schieberegistern für die Ansteuerung einer LED-Matrix ist die im ELV Prototypenadapter-Set PAD4 enthaltene, kaskadierbare 10x7-Dot-Matrix-Anzeige.

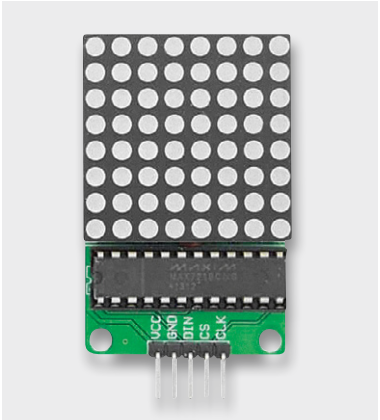

Bild 16: Kaskadierbare LED-Matrixanzeige mit dem Treiberbaustein MAX7219, der eine abgespeckte SPI-Schnittstelle bereitstellt.

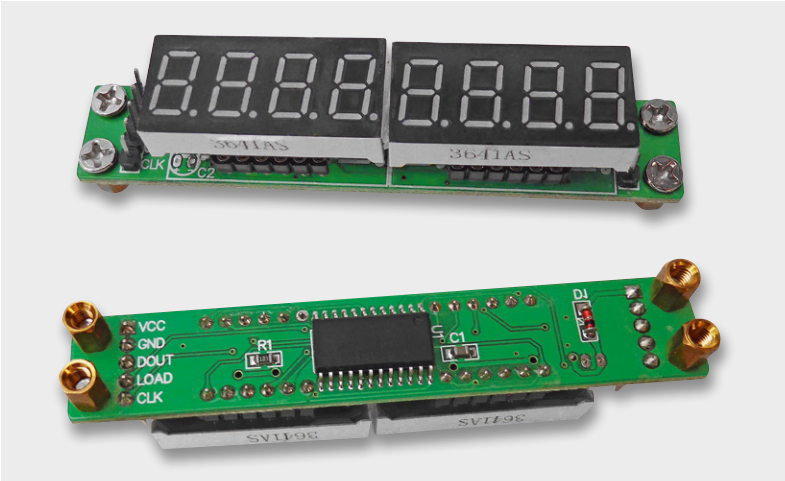

Bild 17: Den MAX7219 findet man auch bei solchen mehrstelligen Segmentanzeigen.

Die Libraries

Auch für alle hier diskutierten Ansteuerarten und Schnittstellen gibt es fertige Libraries. So gibt es für einfache WS2812-Anordnungen z. B. die kompakte Library „light_ws2812". Großer Beliebtheit und gewissermaßen als Standard erfreut sich die „Adafruit_NeoPixel"-Library. Sie ist sehr leistungsstark und extrem universell einsetzbar.

Bei diesen Libraries muss man neben dem Laden der Library per #include nur noch die Anschlussbelegung am steuernden Controller mit (Beispiel Adafruit_Neopixel):

```
#define LED-PIN  x
```

die Anzahl der anzusteuernden Pixel mit:

```
#define LED_COUNT  x
```

die LED-Helligkeit mit:

```
#define BRIGHTNESS x
```

und in der abschließenden Deklaration noch die Reihenfolge der RGBW-Ansteuerung sowie den Bustakt angeben, z. B. so:

```
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRBW +
NEO_KHZ800);
```

Dem folgt schließlich die Initialisierung:

```
void setup() {
  strip.begin();   // INITIALIZE NeoPixel strip object (REQUIRED)
  strip.show();    // Turn OFF all pixels ASAP
  strip.setBrightness(BRIGHTNESS);
}
```

Auch für den MAX7219 gibt es gleich mehrere Libraries. Hier sieht die Start-Sequenz z. B. so aus:

```
#include <max7219.h>

MAX7219 max7219;

void setup() {
  Serial.begin(9600);
  max7219.Begin();
}
```

Für den TM1638 gibt es eine große Anzahl unterschiedlicher Libraries, die je nach Übertragungsverfahren individuelle Initialisierungssequenzen aufweisen, siehe Library-Zusammenstellung. Sowohl für den MAX7219 als auch für den TM1638 halten die jeweiligen Libraries auch entsprechend individuelle Fonts für den Zeichenvorrrat bereit.

Smart-Displays

Wie bereits kurz angesprochen, kann die Darstellung von Texten und Grafiken, insbesondere dynamischen Grafiken, den Mikrocontroller der eigentlichen Anwendung bzw. dessen Speicher erheblich auslasten, so dass für das eigentliche Anwendungsprogramm nur wenig Speicher bleibt und man manchmal eher dazu veranlasst wird, zur Controller-Plattform mit größeren Ressourcen zu greifen.

Außerdem ist der Aufwand, eine ansprechende grafische Anzeige bzw. Bedien- und Anzeigeoberfläche zu programmieren, oft sehr hoch, weil man jede Grafik, jede Linie, jeden Kreis mit mindestens einer Befehlszeile schreiben muss. Und programmieren, Fehlersuche und Simulation kostet viel Zeit und im kommerziellen Bereich auch sehr viel Geld.

Warum also nicht physisch alles, was der Anzeige und ggf. Bedienung gilt, vom Steuerprozessor trennen, sprich, auf das ohnehin nötige Display-Controllerboard verlagern? Das entlastet die Ansteuerung erheblich,weil nur noch rudimentäre Auslöser-Befehle zur Anzeige

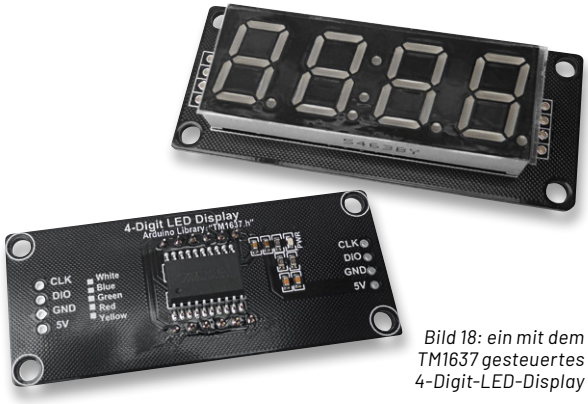

Bild 18: ein mit dem TM1637 gesteuertes 4-Digit-LED-Display

gesendet bzw. Bedienbefehle („Events") von deren Touch-Controller empfangen werden müssen. Der Anwendungsrechner setzt diese Befehle, etwa „On gedrückt" um in eine zuvor zugeordnete Aktion, etwa einen Messwert einzulesen oder einen Aktor über einen Busbefehl, z. B. im CAN-Bus eines Fahrzeugs oder einer Maschine auszulösen.

Die so mit eigener Intelligenz ausgerüsteten Anzeigen nennt man Smart-Displays. Ihr interner Steuerprozessor beherbergt ggf. sogar auf Wechselspeichermedien, z. B. die speicherintensiven Grafikbibliotheken, vorgefertigte Grafiken usw. Diese Displays sind inzwischen preislich gesehen in sehr erschwingliche Bereiche gerückt, herausragende Anbieter sind hier z. B. Nextion, 4D Systems oder Winstar (Bild 19).

Dabei wird die gesamte später abzubildende Anzeige- und Bedienoberfläche, die GUI (Graphical User Interface = Grafische Benutzerschnittstelle) vorab extern in einem zum Display passenden Editor, der auf einem normalen PC läuft, erstellt. Der Anschluss erfolgt dabei meist ganz einfach über eine auf USB umgesetzte serielle UART-Schnittstelle .

Bild 19: Smart-Displays entlasten die Ressourcen des Wirtssystems. Bilder: Nextion, Winstar, 4D Systems

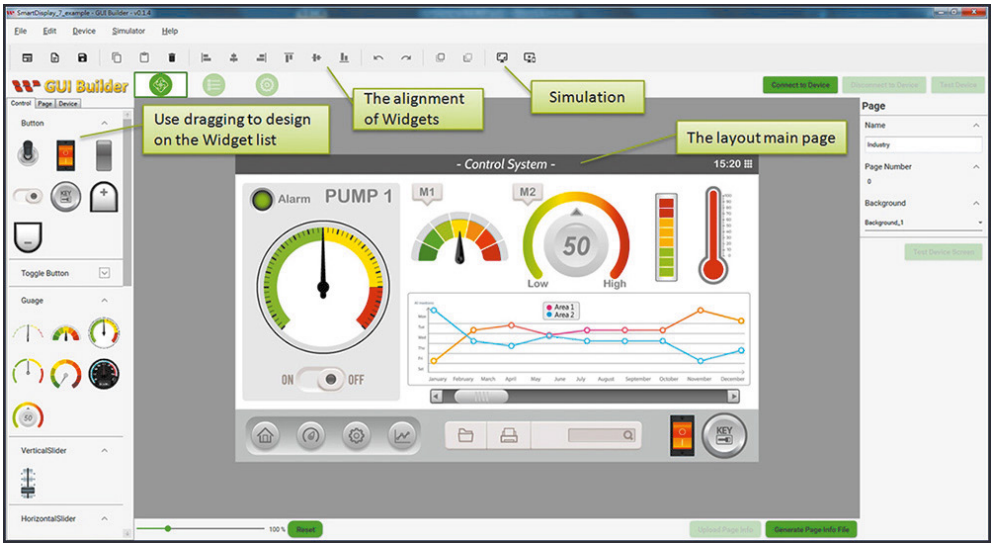

Bild 20: Mit einem GUI-Editor ist eine Displayanordnung für ein Smart-Display einfach und schnell zusammenzustellen. Bild: Winstar

Das Programmieren erfolgt nicht zeilenweise als herkömmliche Programmierung, sondern zumeist als Drag-and-Drop-Zuordnung in der zum geplanten Display konfigurierten Editor-Oberfläche. Das vorgefertigte oder selbst gezeichnete Grafikelement wird hier platziert und es werden die o. a. Beziehungen festgelegt. Ebenso erfolgt die Event-Auswertung eines Touch-Displays. Bild 20 zeigt solch einen Editor, den Winstar Smart Display GUI Builder. In Bild 21 ist eine einfache, mit dem Nextion Editor erstellte Bedienoberfläche für eine E-Auto-Ladestation zu sehen. Die Editoren erlauben auch eine Simulation der Displayfunktionen, so dass sich auch hier der Aufwand begrenzt.

Natürlich senkt diese Technik enorm die Entwicklungszeiten und die Kosten, zudem kann sich der Entwickler mehr seiner eigentlichen Aufgabe, der Anwendung, widmen. ELV

Library-Aufstellung	
LiquidCrystal	https://github.com/arduino-libraries/LiquidCrystal
LiquidCristal I2C	https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library
TFT-Library	https://www.arduino.cc/en/Reference/TFTLibrary
Adafruit GFX	https://github.com/adafruit/Adafruit-GFX-Library
Adafruit SSD1306	https://github.com/adafruit/Adafruit_SSD1306
u8g2	https://github.com/olikraus/u8g2
SSD1306Ascii	https://github.com/greiman/SSD1306Ascii
ssd1306.py	https://github.com/micropython/micropython
Library ePaper	https://github.com/ZinggJM/GxEPD2
Light_ws2812	https://github.com/cpldcpu/light_ws2812
Adafruit_NeoPixel	https://github.com/adafruit/Adafruit_NeoPixel
max7219	https://github.com/JemRF/max7219/
MAX7219	https://github.com/nickgammon/MAX7219
TM1638lite	https://github.com/danja/TM1638lite
tm1638-library	https://github.com/danja/TM1638lite

Bild 21: Eine mit dem Nextion-Editor erstellte Displaygrafik für eine Ladesäule. Bild: Nextion

i Weitere Infos

[1] Arduino-IDE: https://www.arduino.cc/en/software
[2] Java Development Kit: https://www.oracle.com/java/technologies/downloads/
[3] Microsoft Visual Studio: https://code.visualstudio.com/
[4] Scratch: https://scratch.mit.edu/
[5] Python: https://www.python.org/
[6] Microsoft MakeCode: https://www.microsoft.com/de-de/makecode
[7] GNU ARM Embedded GCC: https://xpack.github.io/arm-none-eabi-gcc/install/
[8] I²C-Adress-Scanner von Nick Gammon im Arduino.cc-Playground: https://playground.arduino.cc/Main/I2cScanner/
[9] TFT in der Arduino-IDE: https://www.arduino.cc/en/Reference/TFTLibrary
[10] Tutorial für OLED-Betrieb (Arduino IDE): https://arduinogetstarted.com/arduino-tutorials
[11] Tutorial für TFT am Raspberry Pi: https://tutorials-raspberrypi.de/
[12] Micropython-Treiber für SSD1306: https://github.com/micropython/micropython/blob/master/drivers/display/ssd1306.py
[13] ePaper-Wiki von Waveshare: https://www.waveshare.com/wiki/2.9inch_e-Paper_Module
[14] Library, Examples und Anschluss ePaper: https://github.com/ZinggJM/GxEPD2

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links