Inhaltsverzeichnis

Die Idee

Das Display soll eigenständig arbeiten und dem externen System eine Benutzeroberfläche zur Verfügung stellen. Die Kommunikation erfolg über einfache Strings.

Die Benutzeroberfläche wird im Display erstellt, das externe Gerät liefert nur die Daten. Das Display kann auf der Benutzeroberfläche Schaltflächen definieren, die bei Berührung Meldungen an das externe Gerät senden können.

Aktuell soll das Display zur Darstellung von Wetterdaten aus dem Kurs ***Micropython mit ESP32*** verwendet werden.

Die Hardware

Zur Anzeige wird ein 2.8" TFT Farbdisplay mit 240x320 Pixel verwendet. Die Ansteuerung erfolgt über SPI. Als Controller ist ein ILI9341 eingebaut.

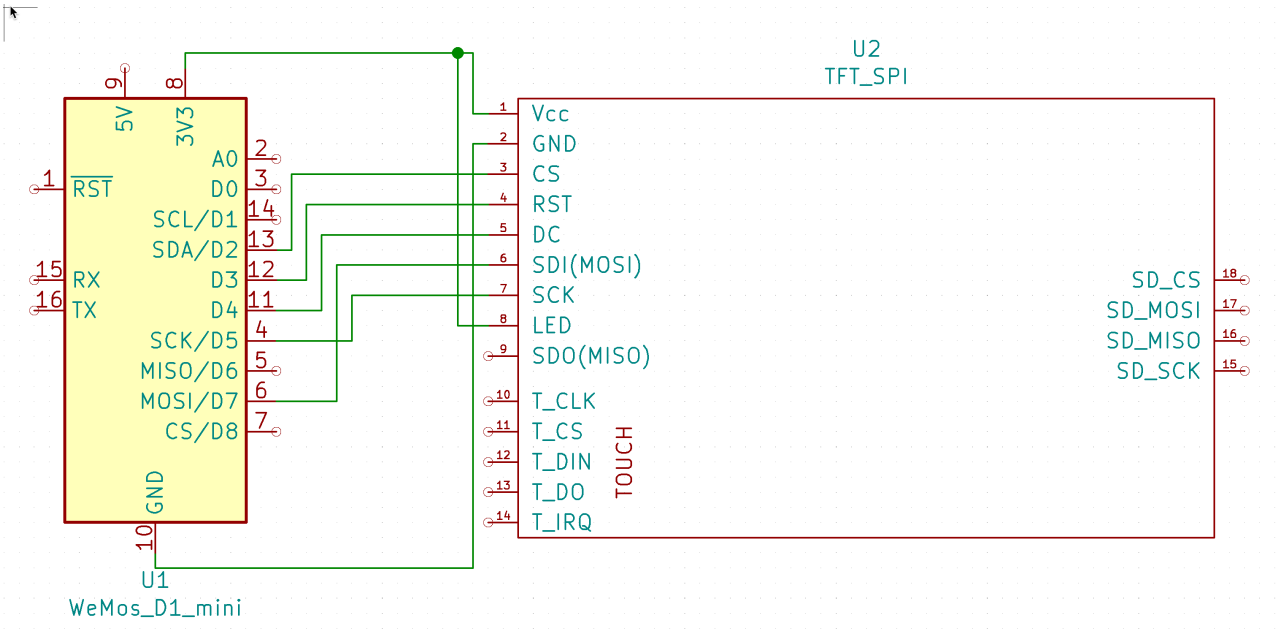
240x320 2.8 "SPI TFT LCD Touch Panel Serial Port Modul Mit PBC ILI9341 2,8 Zoll SPI Serielle weiß LED Display mit Touch Stift|LED-Anzeige| - AliExpress

Zur Steuerung wird ein LOLIN D1 mini Board (ESP8266) verwendet.

https://www.wemos.cc/en/latest/d1/d1_mini.html

In einer erste Stufe wird der Touch noch nicht behandelt, deshalb werden die dazugehörigen Anschlüsse nicht verdrahtet.

Das Display und der Mikrokontroller verwenden 3.3V - Logik, so dass hier keine besonderen Massnahmen notwendig sind.



Display	ESP 8266
VCC	3.3V
GND	GND
LED	3.3V
SCK (SPI Clock)	D5
SDI (MOSI)	D7
DC	D4
RST	D3
CS	D2

Es müssen zwingend die Hardware - SPI - Anschlüsse des ESP8266 verwendet werden (SPI Clock und MOSI). MISO wird nicht verwendet.

Die Software

Die Software wird unter Verwendung einer Display-Bibliothek grössten Teil selbst entwickelt. Dabei werden Hilfsmittel entwickelt, die die Definition der Benutzeroberfläche erleichtern.
Diese Definition erfolgt nicht in einem Designprogramm. Sie erfolgt durch selbstgeschriebenen Code, der aber möglichst einfach sein sollte. Dafür wird die Basis - Firmware im Laufe des Projekts immer mehr ausgebaut.

Entwicklungsumgebung

Wir arbeiten mit Visual Studio Code und PlatformIO und verwenden das Arduino - Framework. In den Videos und den Begleitunterlagen wird die Installation und die Verwendung gezeigt.
Die Informationen sind auch auf GitHub zu finden (ab Lektion 38):
https://github.com/hobbyelektroniker/Arduino_Einstieg

Smartdisplay 1: Erste Versuche: https://youtu.be/wUlyVSPUHeE
Smartdisplay 2: Visual Studio Code und PlatformIO: https://youtu.be/zDvnHF9hlUI
Smartdisplay 3: Eine eigene Klasse: https://youtu.be/w2cHAzAYKWY
Smartdisplay 4: Ein Hauch von SMART:

Grundlegende Bibliothek

Die eigene Software basiert auf der Bodmer - Bibliothek für die ESP - Kontroller und TFT - Displays.

https://github.com/Bodmer/TFT_eSPI

Die Display - Klasse (TFTEsp)

Diese Klasse basiert auf der TFT_eSPI - Klasse von Bodmer und erbt alle ihre Methoden und Eigenschaften. Es gibt pro Projekt nur eine Instanz dieser Klasse. Momentan müssen die Anschlüsse von DC, RST und CS direkt in der Datei User_Setup.h der Bodmer - Library eingetragen werden. Das wird im Video erklärt.

TFTEsp bietet grundlegende Funktionen für das Display an. Einige geerbte Funktionen werden direkt verwendet, teilweise wurden auch neue Funktionen erstellt.

Es muss also zuerst eine Instanz des Displays erstellt werden.

```
TFTEsp disp = TFTEsp();
```

Zur Vorbereitung des Displays wird eine Hintergrund und eine Vordergrundfarbe gesetzt. Zuvor wird es mit begin() gestartet. Wir verwenden das Display im Breitformat (320 x 240 Pixel).

```
disp.begin();
disp.setFgColor(TFT_YELLOW);
disp.setBgColor(TFT_BLUE);
disp.setRotation(1); // Breitformat 320 x 240
disp.cls(); // Bildschirm löschen
```

Zur Farbgebung können in der Bibliothek vorgegebene Farbkonstanten verwendet werden.

```
// Farbdefinitionen aus Library (TFT_eSPI.h)
// #define TFT_BLACK       0x0000      /*   0,   0,   0 */
// #define TFT_NAVY        0x000F      /*   0,   0, 128 */
// #define TFT_DARKGREEN   0x03E0      /*   0, 128,   0 */
// #define TFT_DARKCYAN    0x03EF      /*   0, 128, 128 */
// #define TFT_MAROON      0x7800      /* 128,   0,   0 */
// #define TFT_PURPLE      0x780F      /* 128,   0, 128 */
```

```
// #define TFT_OLIVE       0x7BE0      /* 128, 128,   0 */
// #define TFT_LIGHTGREY   0xD69A      /* 211, 211, 211 */
// #define TFT_DARKGREY    0x7BEF      /* 128, 128, 128 */
// #define TFT_BLUE        0x001F      /*   0,   0, 255 */
// #define TFT_GREEN       0x07E0      /*   0, 255,   0 */
// #define TFT_CYAN        0x07FF      /*   0, 255, 255 */
// #define TFT_RED         0xF800      /* 255,   0,   0 */
// #define TFT_MAGENTA     0xF81F      /* 255,   0, 255 */
// #define TFT_YELLOW      0xFFE0      /* 255, 255,   0 */
// #define TFT_WHITE       0xFFFF      /* 255, 255, 255 */
// #define TFT_ORANGE      0xFDA0      /* 255, 180,   0 */
// #define TFT_GREENYELLOW 0xB7E0      /* 180, 255,   0 */
// #define TFT_PINK        0xFE19      /* 255, 192, 203 */
// #define TFT_BROWN       0x9A60      /* 150,  75,   0 */
// #define TFT_GOLD        0xFEA0      /* 255, 215,   0 */
// #define TFT_SILVER      0xC618      /* 192, 192, 192 */
// #define TFT_SKYBLUE     0x867D      /* 135, 206, 235 */
// #define TFT_VIOLET      0x915C      /* 180,  46, 226 */
```

Momentan verwenden wir ausschliesslich den eingebauten Standardfont. Diese haben 5 Pixel in der Breite und 8 Pixel in der Höhe. Die eigentliche Buchstabenhöhe beträgt 7 Pixel, die unterste Reihe ist für Unterlängen (zum Beispiel beim Buchtaben g) reserviert. Der Abstand zwischen zwei Zeichen beträgt 1 Pixel. Dieser kann in 4 Grössen (1 bis 4) skaliert werden. Wir verwenden standardmässig Grösse 2 (10 x 16 Pixel), manchmal auch Grösse 3.

```
disp.setTextSize(2);
```

Normalerweise sind die Textfarben und die Bildschirmfarben identisch. Sie können aber auch individuell gesetzt werden.

```
setTextColor(uint16_t fgColor, uint16_t bgColor);
```

Zur Ausgabe von Texten können print() und println() verwendet werden. Ihre Funktionsweise entspricht exakt den Funktionen im Serial - Objekt.
Zur Positionierung kann noch vorgängig die Cursorposition gesetzt werden.

```
setCursor(int16_t x, int16_t y);
```

Es handelt sich bei x und y um Grafikkoordinaten. 0 / 0 ist links oben auf dem Bildschirm.
Die Position entspricht immer dem linken oberen Pixel des Zeichens.

Eine direktere Möglichkeit existiert mit drawString().

```
disp.drawString("Ein Text",x,y);
```

Zum Text und dem benötigten Platz lassen sich nützliche Abfragen machen.

`int16_t fontHeight()` gibt die Fonthöhe in Pixel zurück.
`int16_t textWidth("Ein Text")` gibt den Platzbedarf des Textes in Pixel zurück.

Funktionen in TFTEsp

cls()	Clear Screen Füllt den Bildschirm mit der Hintergrundfarbe.	disp.cls();
cls(uint16_t color)	Füllt den Bildschirm mit der angegebenen Farbe color.	disp.cls(TFT_ORANGE);
setFgColor(uint16_t color)	Setzt die Vordergrundfarbe für Bildschirm und Text auf color.	disp.setFgColor(TFT_YELLOW);

setBgColor(uint16_t color)	Setzt die Hintergrundfarbe für Bildschirm und Text auf color.	disp.setBgColor(TFT_BLUE);
setTextColor()	Setzt die Textfarben auf die Farben des Bildschirms.	disp.setTextColor();
setTextColor(uint16_t fgColor, uint16_t bgColor)	Setzt die Textfarben auf die angegebenen fgColor und bgColor. Diese Funktion ruft direkt die entsprechende Funktion der Basisklasse auf.	disp.setTextColor(TFT_YELLOW,TFT_BLUE);
uint16_t TFTEsp::getFgColor() uint16_t TFTEsp::getBgColor()	Abfrage der aktuell gültigen Bildschirmfarben.	
drawText(const char* text, int16_t x, int16_t y, int16_t width)		Schreibt den String text an die Stelle ***x, y***. Vorher wird ein Feld mit der Breite ***width*** und der Höhe des Strings an die Position ***x, y*** gezeichnet. Als Farbe wird ***fgColor*** und ***bgColor*** des Bildschirms verwendet. Der Text wird linksbündig in dieses Feld geschrieben. Wenn kein umfassendes Feld benötigt wird, sollte mit der Funktion ***drawString()*** gearbeitet werden.
drawText(const char* text, int16_t x, int16_t y, int16_t width, char align)		Mit diesem Befehl kann zusätzlich mit ***align*** die Orientierung des Texts angegeben werden. 'l': linksbündig 'c': eingemittet 'r': rechtsbündig
drawText(const char* text, int16_t x, int16_t y, int16_t width, char align, int16_t xPadding, int16_t yPadding)		Hier kann das Feld, das den Text umfasst, vergrössert werden. ***xPadding*** fügt links und rechts die entsprechenden Anzahl Pixel hinzu. ***yPadding*** fügt oben die entsprechende Anzahl Pixel hinzu. Unten wird 1 Pixel weniger hinzugefügt, da die Zeile für die Unterlänge normalerweise ebenfalls frei ist. Die Positionierung x, y bezieht sich immer auf das Feld ohne die Erweiterung durch Padding.

Häufig verwendete Funktionen, die aus der Basisklasse übernommen wurden

begin()	Initialisiert das Objekt. Muss immer vor der ersten Verwendung aufgerufen werden.	disp.begin()
setRotation(uint8_t r)	Ausrichtung des Bildschirms . Wir verwenden normalerweise den Mode 1.	disp.setRotation(1);

setTextSize(uint8_t size)	Setzt die Fontgrösse. Wir verwenden normalerweise den Wert 2.	disp.setTextSize(2);
println(), print(), setTextWrap()	Textausgabe. Wird für Fliesstext verwendet. print() und println() funktionieren analog zu den Funktionen im Serial - Objekt.	disp.println("Hallo");
setTextColor(uint16_t fgColor, uint16_t bgColor)	Setzt die Textfarben auf die angegebenen fgColor und bgColor.	disp.setTextColor(TFT_YELLOW,TFT_BLUE);
setCursor(int16_t x, int16_t y)	Setzt den Cursor auf die angegebene Position. Bei setRotation(1) haben wir 360 Pixel in x-Richtung und 240 Pixel in y - Richtung.	disp.setCursor(25,50);

int16_t drawString(const char *string, int32_t x, int32_t y)
 	
int16_t drawString(const String &string, int32_t x, int32_t y)
	Schreibt einen String direkt an die angegebene Position.
	Funktioniert mit C-Strings und der String - Klasse.
	Es wird jeweils die Breite des Texts zurückgegeben.
		disp.drawString("Position 50 / 100",50,100);
		
		disp.drawString(String("Position 50 / 120"),50,120);

Die Feldklasse TFTField

Diese Klasse dient der Ausgabe von variablen Werten in einem Feld.

TFTField(TFTEsp* tft)	Dem Konstruktor muss ein Pointer auf die aktuelle Display - Klasse mitgegeben werden.	TFTField feuchteFeld = TFTField(&disp);
begin()	Initialisierung das Feld. Wird automatisch aufgerufen.	
setFgColor(uint16_t color); setBgColor(uint16_t color);	Setzt die Farbe des Textes und die Farbe des Hintergrundes. Text und Feld haben immer denselben Hintergrund.	
uint16_t getFgColor(); uint16_t getBgColor();	Die gesetzten Farben können auch abgefragt werden.	
setPos(int16_t x, int16_t y, int16_t width);	Setzt die Position und Breite des Feldes. Die Position bezieht sich auf die linke obere Ecke des Feldes ohne Padding.	
setAlignement(char value)	Orientierung des Textes. 'l': linksbündig 'c': eingemittet 'r': rechtsbündig	
setPadding(int16_t x, int16_t y)	Hier kann das Feld, das den Text umfasst, vergrössert werden. ***x*** fügt links und rechts die entsprechenden Anzahl Pixel hinzu. ***y*** fügt oben die entsprechende Anzahl Pixel hinzu. Unten wird 1 Pixel weniger hinzugefügt, da die Zeile für die Unterlänge normalerweise ebenfalls frei ist.	
void setText(const char* text); void setText(const String text);	Das ist der Text, der geschrieben wird. Das Schreiben erfolgt nicht automatisch. Dazu muss der Befehl ***show()*** ausgeführt werden.	

uint16_t getFieldWidth()	Die Länge des Feldes von der x - Position bis zum Ende des Feldes (inkl. Padding auf der rechten Seite) wird zurückgegeben.
show()	Das Feld und der Text werden angezeigt.

Erstellen einer Benutzeroberfläche

Die Benutzeroberfläche muss programmiert werden.
Wir unterscheiden zwischen statischen und dynamischen Feldern.

Statische Felder werden am Anfang auf den Bildschirm geschrieben und bleiben normalerweise bestehen.
Dynamische Felder bilden Werte aus Variablen ab und können sich laufend ändern.

Bereitstellung der Daten und Objekte

Wir benötigen ein Displayobjekt und pro dynamischem Feld ein Objekt der Klasse TFTField.

```
TFTEsp disp = TFTEsp();       // Das Display

// Das sind die Felder für die anzuzeigenden Werte
TFTField zeitFeld = TFTField(&disp);
TFTField tempFeld = TFTField(&disp);
TFTField feuchteFeld = TFTField(&disp);
TFTField druckFeld = TFTField(&disp);
```

Die anzuzeigenden Daten müssen in Variablen zwischengespeichert werden. Zum Test der Darstellung ist es sinnvoll, den Variablen Testwerte zuzuweisen.

```
// Diese Daten werden von extern empfangen
String zeitWert;
float tempWert, feuchteWert;
int druckWert;

void setTestValues() {
  zeitWert = "TT.MM.JJJJ SS:MM:ss";
  tempWert = 0;
  feuchteWert = 0;
  druckWert = 0;
}
```

In der Funktion prepareScreen() muss zuerst das Display vorbereitet werden.

```
disp.setFgColor(TFT_YELLOW);
disp.setBgColor(TFT_BLUE);
disp.setRotation(1); // Breitformat 320 x 240
disp.cls(); // Bildschirm löschen
```

Jetzt wird ein Titel geschrieben. Er ist auf der obersten Zeile mit einem etwas vergrösserten Font, eingemittet auf der Zeile. Die Position der nächsten Zeile wird auch gleich berechnet.

```
disp.setTextSize(3);
disp.drawText("Wettervorhersage",0,0,disp.width(),'c');
int y = disp.fontHeight() + 10;
```

Auf der nächsten Zeile zeigen wir die Zeit an. Das ist ein dynamisches Feld, das mit einer Hintergrundfarbe versehen wird. Auch hier kann gleich die Position der nächsten Zeile ermittelt werden.

```
// Zeitfeld
disp.setTextSize(2);
```

```
zeitFeld.setAlignement('c');
zeitFeld.setPos(0,y,disp.width());
zeitFeld.setPadding(0,2);
zeitFeld.setBgColor(TFT_BROWN);
y += disp.fontHeight() + 30;
```

Es sollen drei Felder für Temperatur, Feuchte und Luftdruck untereinander dargestellt werden. Die Werte sollen rechtsbündig untereinander stehen. Dazu berechnen wir passende Feldgrössen.

```
int feldAbstand = disp.fontHeight() + 8;
int labelBreite = disp.textWidth("Temperatur"); // Längster Text
int feldtextBreite = disp.textWidth(" 1025");
```

Dann werden die statischen Texte geschrieben und die Positionen der dynamischen Felder festgelegt.

```
// Temperatur
int x = 0;
disp.drawString("Temperatur",x,y);
tempFeld.setAlignement('r');
tempFeld.setPos(labelBreite+10,y,feldtextBreite);
disp.drawString("C",x+labelBreite+10+tempFeld.getFieldWidth()+5,y);
y += feldAbstand;

// Feuchte
x = 0;
disp.drawString("Feuchte",x,y);
feuchteFeld.setAlignement('r');
feuchteFeld.setPos(labelBreite+10,y,feldtextBreite);
disp.drawString("%",x+labelBreite+10+feuchteFeld.getFieldWidth()+5,y);
y += feldAbstand;

...
```

Mit dieser Definition werden die statischen Elemente geschrieben, die dynamische Felder werden aber noch nicht angezeigt..

Ausgabe von Werten

Wir nehmen an, dass die Messwerte als Zahlen vorliegen. Der Zeitwert ist ein String, der Datum und Zeit beinhaltet. Alle Werte werden zu Strings umformatiert und den dynamischen Feldern übergeben. Am Schluss müssen wir die Felder nur noch anzeigen.

```
// Die variablen Werte werden angezeigt
void showValues() {
  tempFeld.setText(String(tempWert,1));
  feuchteFeld.setText(String(feuchteWert,1));
  druckFeld.setText(String(druckWert));
  zeitFeld.setText(zeitWert);
  zeitFeld.show();
  tempFeld.show();
  feuchteFeld.show();
  druckFeld.show();
}
```

Bezug der Werte

Die Werte möchten wir aus einer externen Quelle beziehen. Diese werden von unserem ESP32 - Board an uns gesandt und wir sollen sie nur noch anzeigen. Das geschieht über einen String, den uns die Datenquelle liefert. Dieser enthält Messwerte und Kommandos.

Wir stellen dazu eine Funktion newCmdOrValue() zur Verfügung. Momentan haben wir noch keine Kommunikation, daher erstellen wir die Aufrufe direkt in unserem Programm.

```
newCmdOrValue("[temp]=24");
newCmdOrValue("[feuchte]=43.5");
newCmdOrValue("[druck]=1029");
newCmdOrValue("[zeit]=11.01.2021 16:50:35");
newCmdOrValue("<show>");
```

Jetzt ist es Zeit, das Datenformat anzuschauen. Im Moment ist es noch sehr einfach. Es kennt keinen Rückkanal und kann nur Texte und Werte darstellen.

Es gibt zwei grundsätzlich unterschiedliche Formate: Befehle und Daten.

Befehle beginnen immer mit < und enden mit >. Dazwischen steht das Schlüsselwort. Im Moment sind nur zwei definiert.

<cls>	Bildschirm löschen und neu aufbauen, dann Werte anzeigen
<show>	Werte anzeigen

Daten werden als Wertepaar aus Schlüssel und Wert übertragen.

[dataKey]=value	dataKey bezeichnet die Variable, value entspricht dem Wert

Die Decodierung dieser Strings müssen wir programmieren.

```
// Ein neuer Wert wird der entsprechenden Variablen zugewiesen
void newValue(String key, String val) {
  if (key.equals("temp")) {
    tempWert = val.toFloat();
  } else if (key.equals("feuchte")) {
    feuchteWert = val.toFloat();
  } else if (key.equals("druck")) {
    druckWert = val.toInt();
  } else if (key.equals("zeit")) {
    zeitWert = val;
  }
}

void newCmdOrValue(const char* text) {
  String cmd = String(text);
  if (cmd.equals("<cls>")) {
    prepareScreen();
    showValues();
  } else if (cmd.equals("<show>")) {
    showValues();
  } else if (cmd.startsWith("[")) {
    int i = cmd.indexOf("]=");
    if (i<0) return;
    String key = cmd.substring(1,i);
    String val = cmd.substring(i+2);
    newValue(key,val);
  }
}
```

Für den Test erlauben wir die Eingabe der Befehle über den seriellen Monitor.

```
void fromSerial() {
  Serial.println("Daten eingeben");
  char buffer[80];
  int anzahl = 0;
  char ch;

  while (true) {
    if (Serial.available()) {
      ch = Serial.read();
      if (ch == 13 || anzahl == 79) { // String beendet
        buffer[anzahl] = 0;
        newCmdOrValue(buffer);
        anzahl = 0;
        Serial.println();
      }
      if (ch >= ' ') {  // keine Steuerzeichen einfügen
        buffer[anzahl++] = ch;
        Serial.print(ch);
      }
    }
  }
}
```

Datenbezug von einem externen Projekt

Serielle Schnittstelle

Der ESP8266 hat nur eine UART. Diese wird normalerweise für die Programmierung des Chips über USB verwendet. Ausserdem ist darüber eine Kommunikation mit dem seriellen Monitor möglich.
Es ist möglich, diese UART ebenfalls für die Kommunikation zu einem externen Board zu nutzen.

Wir haben also drei Nutzungsarten, die nebeneinander funktionieren sollen. Das funktioniert gut, solange sie nicht gleichzeitig genutzt werden. In der Praxis bedeutet das, dass wir die Verbindung zum externen Board trennen, bevor wir den ESP8266 programmieren. Auf die Nutzung des seriellen Monitors verzichten wir möglichst.

Die Boards werden wir folgt verbunden:

ESP 32	**ESP 8266**	
GND	GND	GND bleibt ständig verbunden
GPIO 15, tx	rx	Diese Verbindung muss zum Programmieren des ESP8266 unterbrochen werden.
GPIO 2, rx	tx	Wird momentan nicht verbunden, da wir noch keinen Rückkanal verwenden.

Die Datenquelle

Als Datenquelle dient das ESP32 - Board aus der Videoreihe Micropython mit ESP32. Wir gehen vom Programm aus Lektion 19 aus. Wir verwenden UART 2 zur Kommunikation, UART 0 und 1 sind intern belegt. Die UART 2 lässt sich auf beliebige freie Pins mappen. In unserem Beispiel verwenden wir 2 für rx und ***15 für tx***. rx wird momentan nicht verwendet.

In Micropython verwenden wir ein ***UART***-Objekt, das im Module ***machine*** zu finden ist.

```
import machine
uart = machine.UART(2,9600,rx=2,tx=15)
```

Daten können mit

```
uart.write("<show>\r")
```

gesendet werden.

Der Datenempfang

Der Datenempfang erfolgt über die einzige UART des ESP8266. Dazu wird wie beim seriellen Monitor die Serial - Klasse benutzt.

```
Serial.begin(9600);
if (Serial.available()) {
  ch = Serial.read();
  ..
```

Details zur Software findest du im Dokument ***Zusammenfassung Programmierung.pdf***.