Az-Touch Mod als analoge Uhr

Bevor ich größere Projekte angehe, in meinem Fall eine vollautomatische Aquariumsteuerung mit einigen Features, versuche ich Teilprojekte daraus zu machen und später das große Ganze daraus zu programmieren. Das hat viele Vorteile, da man Code schon vorher testen und optimieren kann, bevor man mit dem großen Ganzen beginnt. Dieses Projekt entstand, da ich zunächst die Daten für Sonnenaufgang und – untergang, sowie die Zeitzone abfragen wollte. Mit diesen doch geringen Daten, kam direkt die Idee auf, eine analoge Uhr auf dem Az-Touch Wandmod zu realisieren, welche auch in Teilen dann bei der Aquariumsteuerung eingesetzt wird. Gerade das Az-Touch Wandmod eignet sich mit einigen Modifikationen perfekt für solche Aufgaben. Einige Zeilen Code habe ich schon in anderen Projekten verwendet und kann hier eine Optimierungsschleife durchlaufen. Damit die Uhr aber nicht das Einzige ist, wird auch die tagesaktuelle Zeit von Sonnenaufgang und -untergang auf dem Display visualisiert. Damit wir aber nicht nur einfache Text anzeigen, sollen monochrome Bilder auf dem Display genutzt werden. Wie das geht und wie man z.B. Bitmaps in für den ESP32 NodeMCU lesbare Daten umwandelt soll heute das Thema vom Blogbeitrag werden.

Die Hard- und Software für diesen Blog

Damit Sie diesen Blog nachbauen können, brauchen Sie die Komponenten aus Tabelle 1:

Pos	Anzahl	Bauteil
1	1	ESP32 NodeMCU Module WLAN WiFi Development Board mit CP2102 (Nachfolgermodell zum ESP8266)
2 oder	1	AZ-Touch MOD Wandgehäuseset mit 2,4 Zoll Touchscreen für ESP8266 und ESP 32
3	1	AZ-Touch MOD Wandgehäuseset mit 2,8 Zoll Touchscreen für ESP8266 und ESP32
4 (optional)	1	12V Netzteil zur Spannungsversorgung

Tabelle 1: Bauteile für die analoge Uhr

An Software brauchen Sie:

- Arduino IDE (https://www.arduino.cc/en/Main/Software), hier am besten die aktuellste Version herunterladen
- Die Bibliothek TFT_eSPI, NTPClient, ArduinoJson, WiFi, TimeLib mit allen Abhängigkeiten.
- Das Programm LCD Image Converter (https://sourceforge.net/p/lcd-image-converter/activity/?page=0&limit=100#604634eaa7028b0bbaf4db72) in der Version 20190317

Zusätzlich benötigen Sie einen Zugang zu openweathermap.org, wobei die kostenlose Basisversion für dieses Projekt vollkommen ausreicht.

Das Programm LCD Image Converter

Wenn Programme entwickelt werden, so muss man immer darauf achten, dass die Ausgaben auf der Displayanzeige jeder versteht. Im Falle von Diagnoseausgaben reicht Englisch oder die Muttersprache, aber bei grafischen Benutzeroberflächen für den Endanwender, kurz GUI, muss entweder alles in jede Sprache übersetzt werden oder man bedient sich eindeutigen Piktogrammen.
Für meine analoge Uhr wollte ich nun nicht alles ausschreiben, gerade auch durch den limitierten Platz, sondern Piktogramme für den Sonnenaufgang und -untergang benutzten, siehe Abbildung 1.

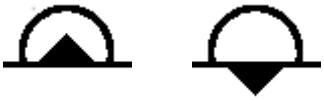

Abbildung 1: Bitmaps für analoge Uhr

Das Problem ist, die Bilder habe ich mit Paint schnell als Bitmap, Endung bmp, erstellt, diese kann das Az-Touch bzw. der ESP32 NodeMCU aber nicht lesen. Da es sich noch um einfache Piktogramme handelt, könnte nun im Quellcode eine Funktion geschrieben werden, die mit Zeichenfunktionen, Kreise, Linien, etc., diese auf das Display zeichnet, aber das ist viel zu viel Mathematik und ein Bild kann schnell in lesbaren Maschinencode umgewandelt werden.

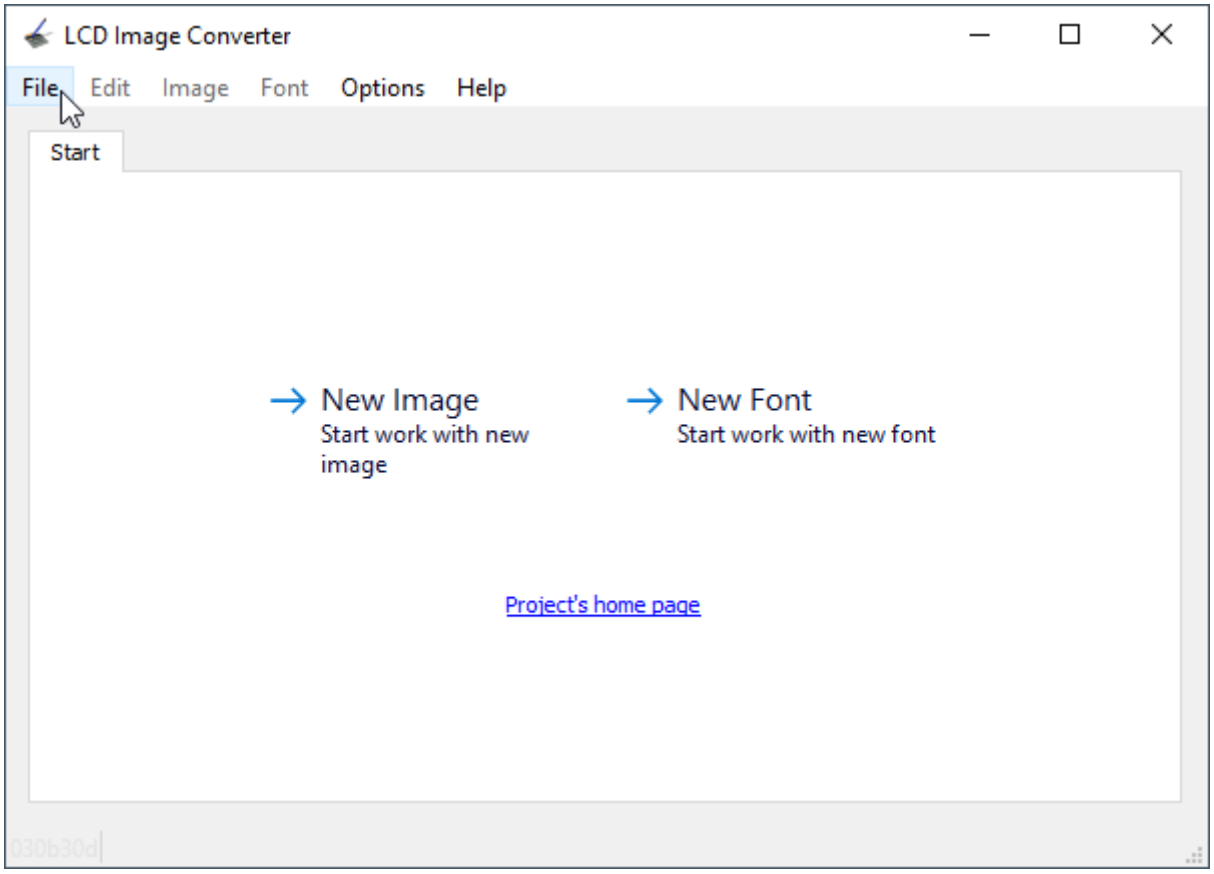

Abbildung 2: Das Programm LCD Image Converter

Meine Wahl fiel auf das Programm LCD Image Converter, siehe Abbildung 2, welches sowohl ein sehr einfacher Zeichen- und Fonteditor ist, aber auch die Möglichkeit bietet Bilder in Maschinencode zu konvertieren. Letzteres werde ich Ihnen für monochrome Bilder zeigen, kann aber auch für farbige Bilder angewendet werden.

Zunächst muss über File -> Open das zu konvertierende Bild ausgewählt werden, siehe Abbildung 3.

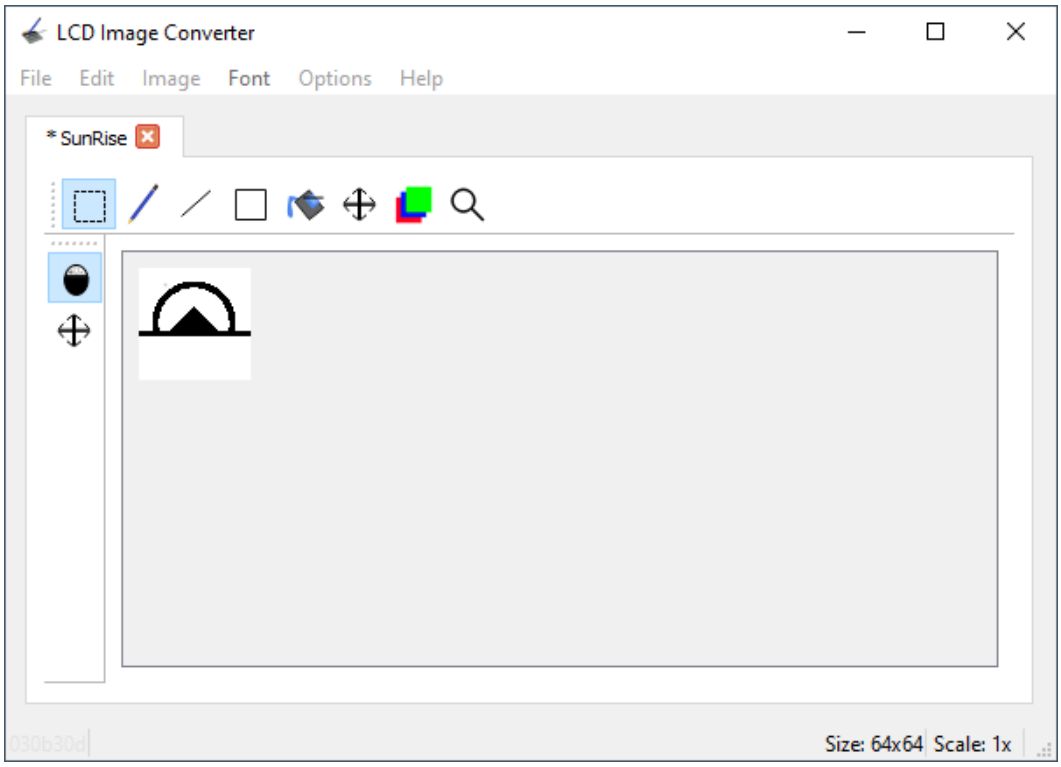

Abbildung 3: Eine Bilddatei laden

Nach dem laden vom Bild haben Sie gleich zwei interessante Änderungen in der Oberfläche. Zunächst sehen Sie das ausgewählte Bild in der Mitte und unten rechts die Bildgröße, hier 64 x 64 Pixel.
Als nächstes öffnen wir über Options -> Conversion… das Conversion-Interface aus, siehe Abbildung 4.

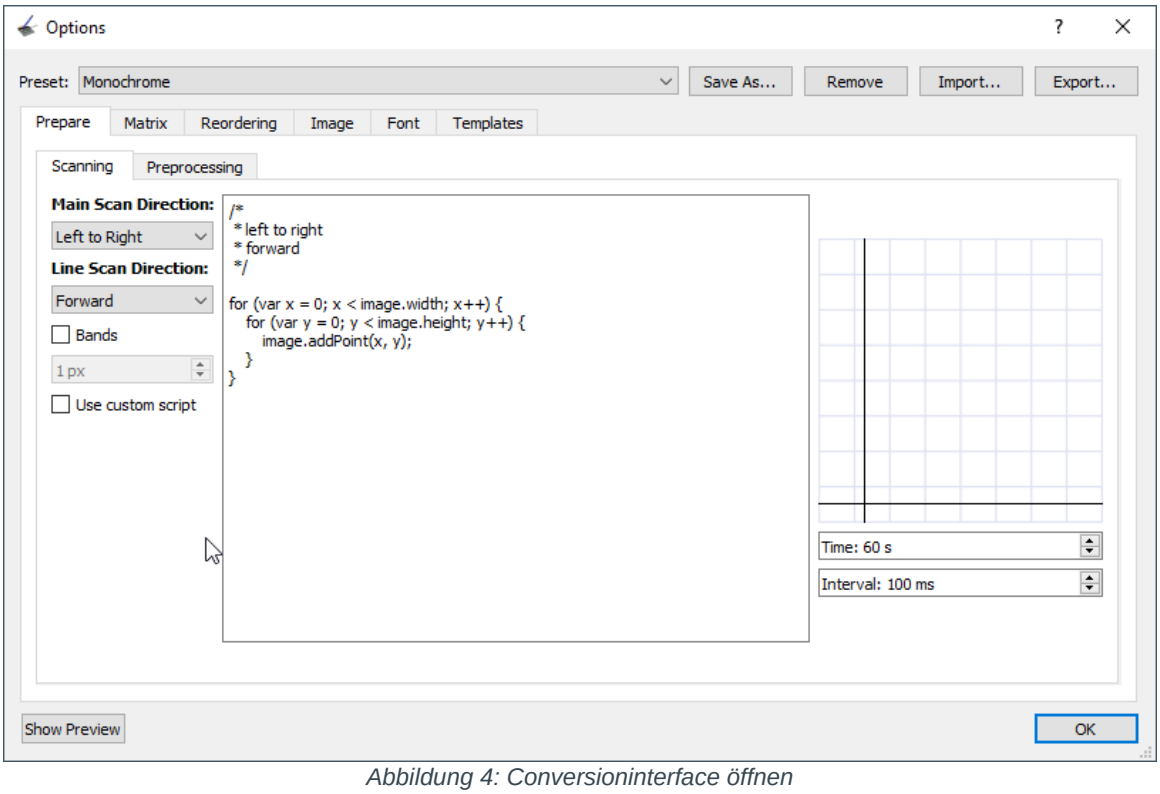

Abbildung 4: Conversioninterface öffnen

Damit das Bild auf dem Display korrekt angezeigt wird, in unserem Fall ein monochromes Bild, müssen noch die Einstellungen angepasst werden. Zunächst wird das Profil auf **Monochrome** gestellt und im Reiter „Scanning“ **Top to Buttom**, siehe Abbildung 5.

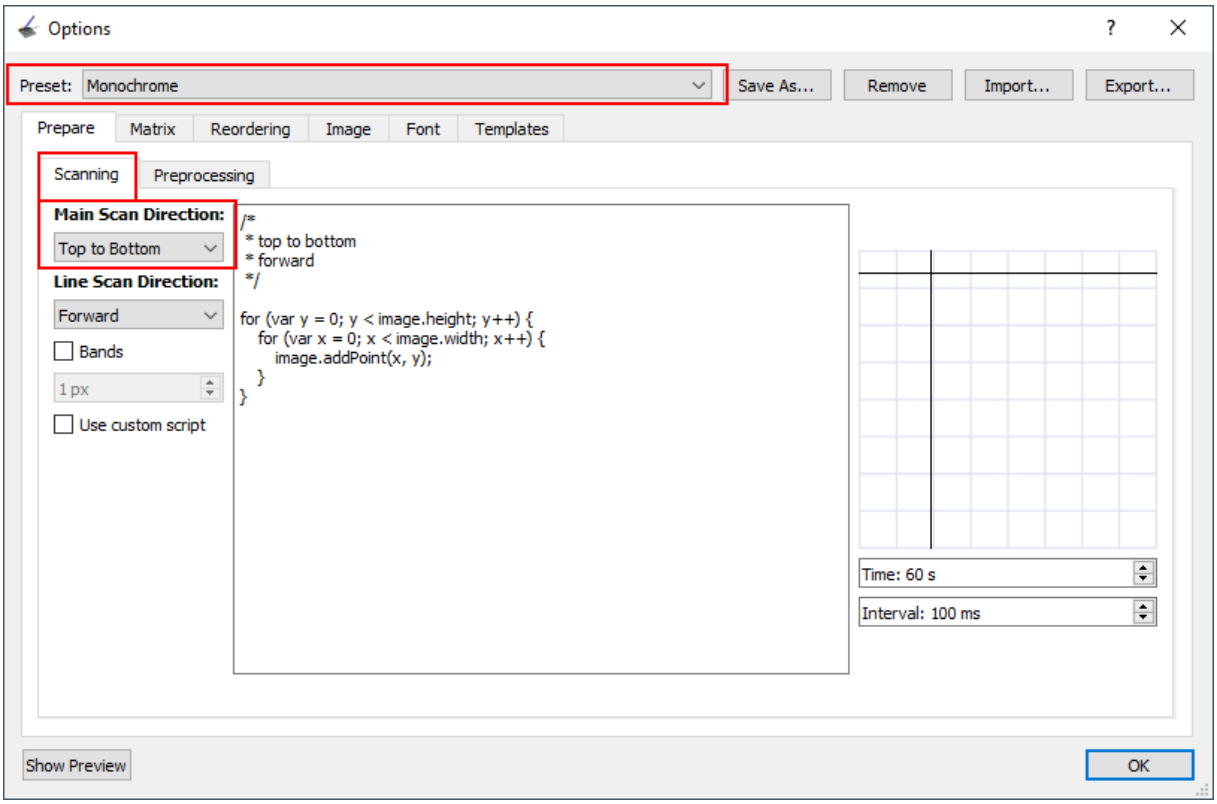

Abbildung 5: Scanning-Einstellungen anpassen

Das bewirkt, das der Maschinencode die korrekte Orientierung hat, von oben nach unten. Als nächstes wird in den Reiter Preprocessing gewechselt, siehe Abbildung 6.

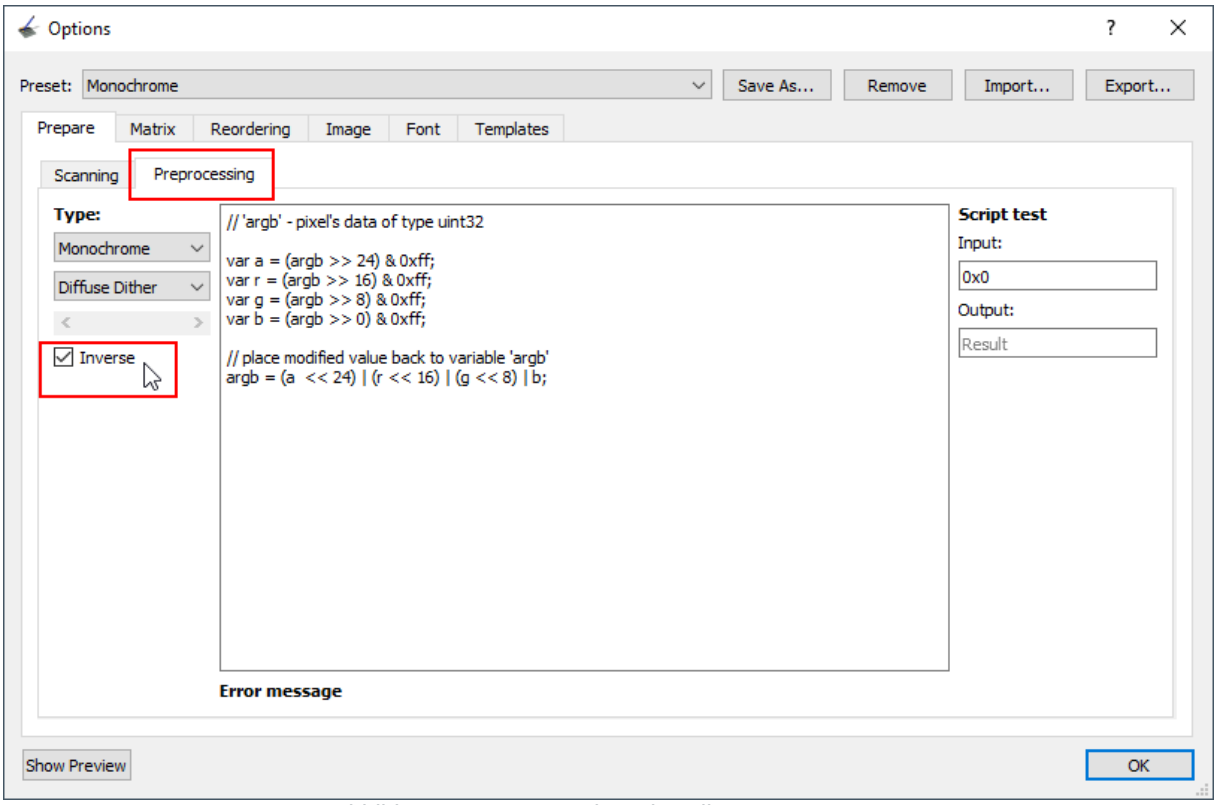

Abbildung 6: Preprocessing-Einstellungen anpassen

Hier wird die Option **Inverse** aktiviert, da sonst nicht das Bild später in der gewählten Farbe, sondern der Hintergrund vom Bild auf dem Display eingefärbt wird. Nun noch den Button **Show Preview** drücken und das Ergebnis wird im nächsten Fenster angezeigt, siehe Abbildung 7.

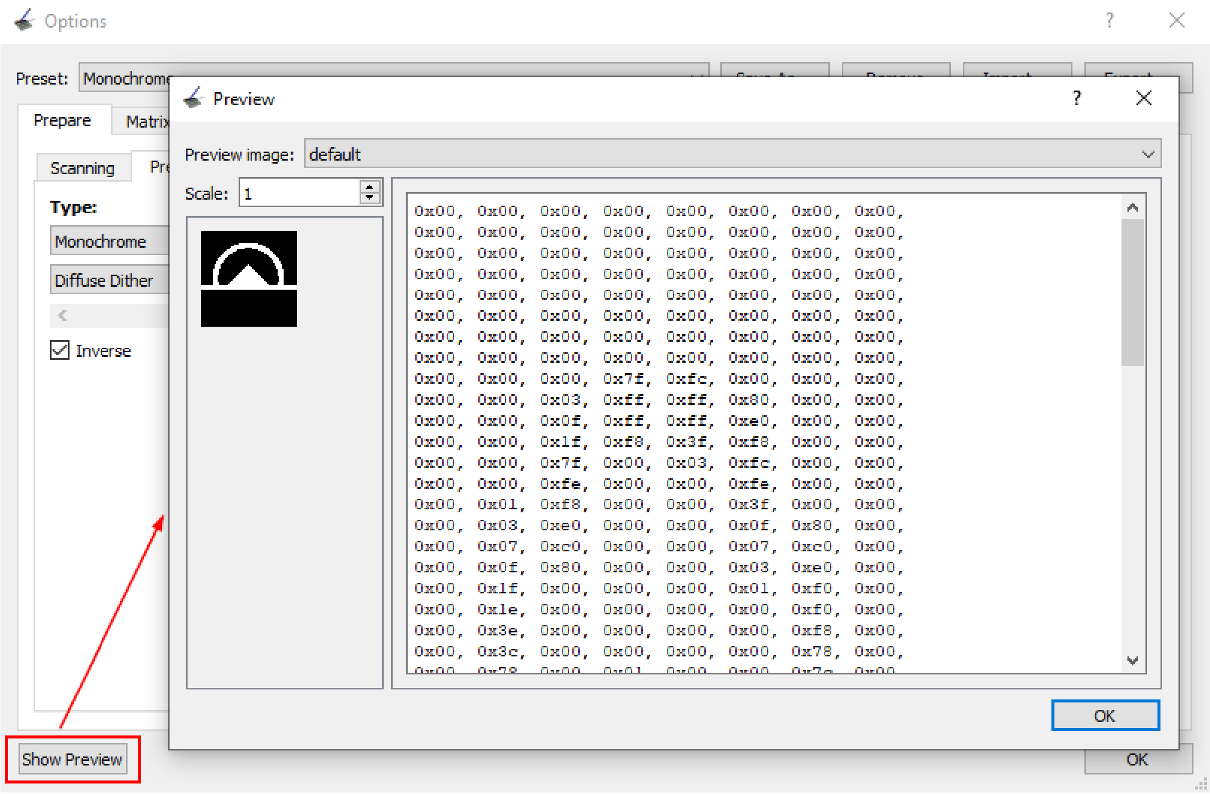

Abbildung 7: Voransicht des Bildes und Maschinencode

Interessant ist hier der rechte Teil im neuen Fenster **Preview**, bei der die vielen Hex-Daten angezeigt werden. Hierbei handelt es sich nun um die Bilddaten in Maschinencode, was später auf dem Az-Touch angezeigt werden soll. Kopieren Sie den gesamten Inhalt und erstellen Sie eine neue Headerdatei, z.B. **bitmap.h**.
Fügen Sie zunächst die folgenden Zeilen am Anfang der Datei ein, siehe Code 1.

```
#ifndef _BITMAP_H
#define _BITMAP_H
#endif

#ifdef ARDUINO_ARCH_AVR
#include <avr/pgmspace.h>
#else
#include <pgmspace.h>
#endif
```

Code 1: Defines für die bitmap.h

Die ersten 3 Zeilen bewirken, dass die Headerdatei richtig im Code hinterlegt wird. Die weiteren Zeilen bewirken, dass die pgmspace.h für ESP-MicroController oder Arduino-MicroController korrekt geladen werden.
Damit nun im späteren der Quellcode das monochrome Bild **sunrise** genutzt werden kann, muss die entsprechende Variable angelegt werden, siehe Code 2.

```
const unsigned char sunrise[] PROGMEM = {

};
```

Code 2: Array für monochromes Bild anlegen

Den kopierten Inhalt vom LCD Image Converter fügen Sie nun dem Array hinzu, siehe Code 3, was den gekürzten Inhalt der neu angelegten Header-Datei wiedergibt.

```
#ifndef _BITMAP_H
#define _BITMAP_H
#endif

#ifdef ARDUINO_ARCH_AVR
#include <avr/pgmspace.h>
#else
#include <pgmspace.h>
#endif

//Icon for sunset 64*64 pixel
const unsigned char sunrise[] PROGMEM = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
....
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

Code 3: Maschinencode für das Array sunrise, stark gekürzt

Das gleiche Vorgehen mit dem Array, wird nun für alle weiteren Grafiken durchgeführt. Am Ende haben Sie, je nach Anzahl von Grafiken, eine Header-Datei mit vielen Zeilen. Für die zwei Grafiken in diesem Projekt, fallen knapp 150 Zeilen Code an.
Damit sind die Vorbereitungen für die analoge Uhr abgeschlossen.

Die analoge Uhr auf dem Az-Touch

Insgesamt umfasst dieses kleinere Projekt knapp 500 Zeilen Code. Der Code liegt wie gewohnt auf meinem Github-Repository für Sie bereit und ist mit vielen Kommentaren versehen, was auch ein Grund für die vielen Zeilen sind. Jedoch sollen einige Funktionen und Abschnitte näher erläutert werden, damit die Sie verstehen, wie der Code funktioniert. Ganz zu Anfang im Quellcode werden die Defines für die Farben der Uhr festgelegt, siehe Code 4.

```
/* --- Some color definitions, feel free to change --- */
#define FACE           TFT_ORANGE
#define NUMERIC_POINT   TFT_BLACK
#define BACKGROUND      TFT_WHITE
#define HOURCOLOR       TFT_RED
#define MINUTECOLOR     TFT_BLACK
```

Code 4: Defines für die Farben der Uhr

Hier können Sie nach Lust und Laune die Farbe tauschen, achten Sie dazu auf die Defines in der Datei **TFT_eSPI.h** in der gleichnamigen Bibliothek. Das vereinfacht den Austausch der Farben ungemein.

Nach den Includes der benötigten Bibliotheken kommt einer der wichtigsten Parts für das Projekt, das Setzen der WLAN-Paramter und die benötigten API-Daten für openweathermap.org, siehe Code 5.

```
const char* ssid       = "";       //CHANGE SSID from network
const char* password   = "";  //CHANGE Password from network

/*--- Needed variables for openweathermap.org ---*/
const String apiKey = ""; //CHANGE API-Key from openweathermap.org
const String location = "";                 //CHANGE Location for openweathermap.org
```

Code 5: WLAN-Zugang und API-Informationen openweathermap.org

An dieser Stelle fügen Sie bitte die benötigen Daten ein, damit später der ESP32 NodeMCU sich mit dem Internet verbinden und auch die Wetterdaten von openweathermap.org runterladen kann. Hier sollten Sie sich schon bei openweathermap.org registriert haben.

Die setup()-Funktion aktiviert das TFT, das WLAN mit den hinterlegten Daten, synchronisiert die Zeit vom ESP32 NodeMCU mit dem NTP-Server und zeichnet die Basisgrafiken auf das Display. Damit Sie sehen, welcher Teil der setup()-Funktion gerade bearbeitet wird, gibt es eine Art Statusausgabe auf dem Display, siehe Abbildung 8.

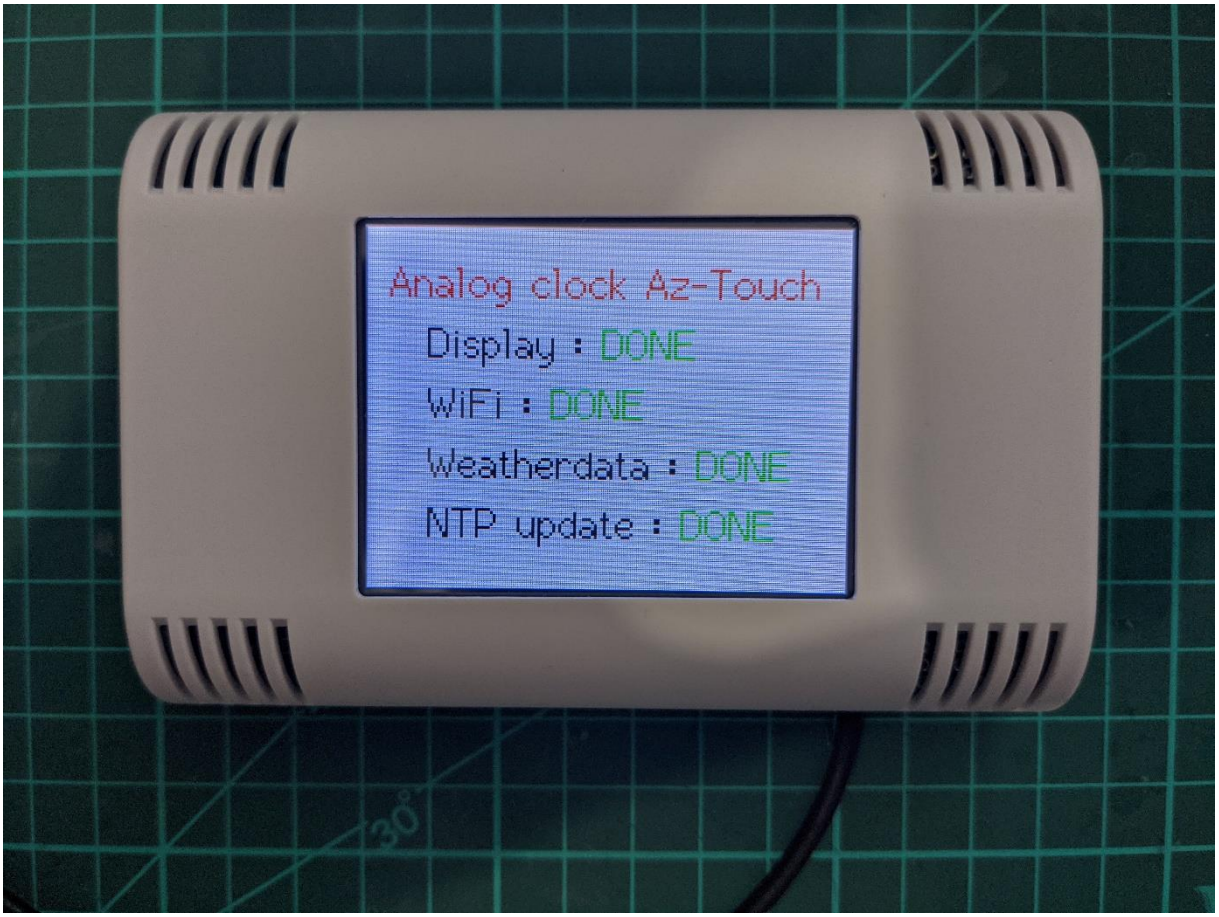

Abbildung 8: Status vom Start des Programms

Damit sehen Sie dann auch sofort, wo das Programm ggf. nicht weiterkommt und Sie die analoge Uhr nicht sehen können. Das hilft ungemein, wenn der Az-Touch mal nicht am PC angeschlossen ist. Ist alles erfolgreich abgeschlossen, wird nach 5 Sekunden die Uhr mit allen Daten angezeigt, siehe Abbildung 9.

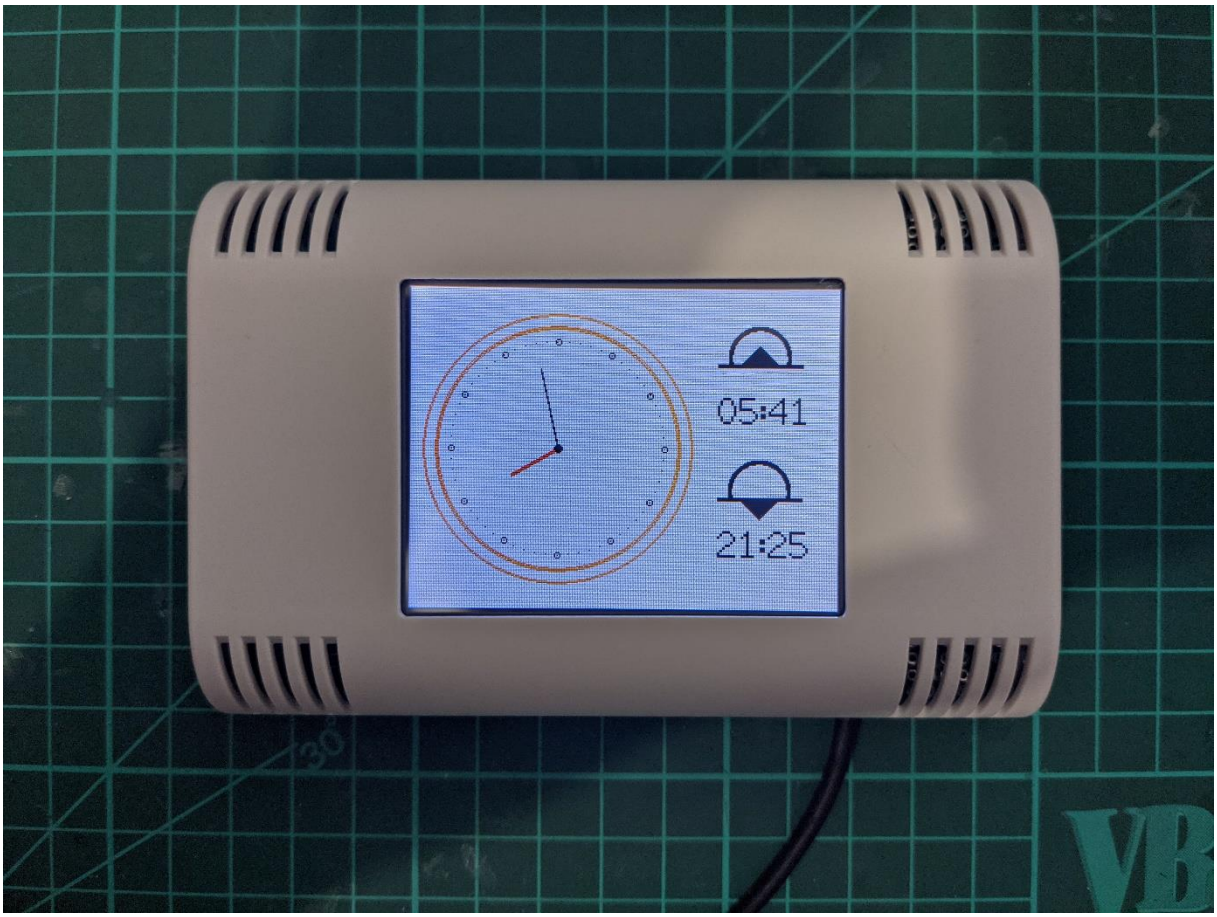

Abbildung 9: Az-Touch mit allen Daten

Wie Sie schnell sehen werden, passiert in der loop()-Funktion vom Umfang her nicht viel, siehe Code 6. Lediglich vier benötigten Funktionen werden aufgerufen und Daten an den seriellen Monitor geschickt.

```
/*
* ================================================================
* Function:     loop
* Returns:      void
* Description:  Main loop to let program work
* ================================================================
*/
void loop()
{
  DrawTime();
  UpdateOpenWeatherMapData();
  UpdateNTPTime();
  UpdateSunriseSunset();

  //Serial output
  if(pPrevSecond != second() || bFirstrun)
  {
    Serial.print(daysOfTheWeek[weekday()] + ", ");
    Serial.print(GetDigits(hour()) );
    Serial.print(":");
    Serial.print(GetDigits(minute()) );
    Serial.print(":");
    Serial.println(GetDigits(second()) );
```

```
    pPrevSecond = second();
  }

  if(bFirstrun)
    bFirstrun = false;
}
```

Code 6: Die loop()-Funktion des Projekts

Aber genau hier liegt der Vorteil! Die loop()-Funktion ist nicht überfüllt und man kann schnell Code austauschen. Die Funktionen haben eindeutige Namen, weswegen Sie schnell herausfinden werden, welche Funktion welchen Arbeitsschritt leistet. Jedoch möchte ich an dieser Stelle auf alle Funktionen der loop()-Funktion eingehen.

DrawTime() bewirkt, dass die Zeiger der Uhr bei jedem Stunden- oder Minutenwechsel neu gezeichnet werden. Wenn Sie die Funktion im Quellcode näher ansehen, wird zuerst der alte Zeiger mit der Hintergrundfarbe vom Display überschrieben und danach der neue Zeiger gezeichnet. Hier steckt für die Zeiger die meiste Mathematik, da entsprechend der Uhrzeit die Zeiger im richtigen Winkel angeordnet sein müssen. Da im Quellcode nicht jeder Pixel einzeln von uns gezeichnet wird, sondern eine Linie mit einem Start- und Endpunkt definieren wird, müssen wir über die Winkelfunktionen Sinus und Cosinus, ausgehend vom Uhrenmittelpunkt, den Endpunkt der Zeiger errechnen. Ist dies geschehen, wird mittels drawLine() der Zeiger in die Uhr gezeichnet.
Scheuen Sie nicht, sich die Funktion DrawTime() und die darin aufgerufenen Funktionen mal genauer anzusehen, auch wenn das Endergebnis funktioniert.

Direkt nach DrawTime() folgt die Funktion UpdateOpenWeatherMapData(), welche, wie der Name schon sagt, die aktuellen Daten von openweathermap.org anfragt. Dies geschieht in drei wesentlichen Schritten:

1. Anfrage für den Client vorbereiten und abschicken
2. Antwort abwarten und benötigten Daten, hier das JSON-Format, vorbearbeiten
3. Daten vom JSON-Format aufbereiten und wichtige Daten speichern

Dieser Teil vom Code ist etwas komplexer, da ein https-Telegramm für openweathermap.org vorbereitet werden, hier über die Funktion RequestData() und die Antwort korrekt formatiert werden muss. Dazu sollten beim Antworttelegramm unnötige Daten eliminiert werden. Im Anschluss konvertiert UpdateOpenWeatherMapData() den empfangenen RAW-String in JSON um und extrahiert die Daten für Zeitzone, Sonnenaufgang und -untergang. Diese neuen Daten werden mit den vorhandenen verglichen und bei einer Änderung ein Update der GUI oder der Zeitzone angestoßen. Da unsere benötigten Daten sich nicht sekündlich ändern, wird die oben beschriebene Arbeit nur alle 30 Minuten durchgeführt.
Damit unsere analoge Uhr auch immer die richtige Zeit anzeigt, wird mittels UpdateNTPTime()-Funktion alle 15 Minuten die interne Uhr vom ESP32 NodeMCU mit dem NTP-Server synchronisiert, somit sollte unsere Uhr immer sekundengenau laufen.

Zuletzt kommt noch die UpdateSunriseSunset()-Funktion, die im Gegensatz zu den bisherigen Funktionen wahrscheinlich die einfachste ist. Sollte sich durch die UpdateOpenWeatherMapData()-Funktion eine Änderung von Sonnenaufgang und -untergang ergeben haben, so wird die alte Uhrzeit auf dem Display gelöscht und die neue Zeit eingetragen.

So einfach das nun wahrscheinlich alles in der Theorie klingen mag, gerade die ersten drei Funktionen in der loop()-Funktion sind doch recht komplexer Natur. Zum einen, weil hier viel Mathematik genutzt wird, zum anderen weil die Beschaffung der Daten von openweathermap.org nicht trivial ist. Eine große Hilfe bei der Beschaffung der Daten war die gute und ausführliche API-Doku von openweathermap.org und die Tatsache, dass ich eine ähnlichen RequestData()-Funktion schon einmal für ein anderes Projekt erstellt habe. Diese sollte fast universell nutzbar sein, sofern Sie für ein anderes Projekt von einer anderen Seite

Daten anfordern wollen. Da ich kein Freund von http-Übertragungen bin, nutzt die RequestData()-Funktion die sicherere https-Variante, welche aber auch für http eingesetzt werden kann.

Vorbereitung für das Projekt

Bevor Sie nun das Programm kompilieren und danach frustriert sind, dass dieses fehlschlägt. Im Projektverzeichnis finden Sie eine Datei mit dem Namen **User_Setup.h**. Diese ersetzen Sie bitte im Order C:\User**IHR-Name**\Documents\Arduino\libraries\TFT_eSPI, siehe Abbildung 10. Es macht Sinn, zuvor die dort befindliche Datei mit gleichem Namen umzubenennen, z,B. in **User_Setup_old.h**.

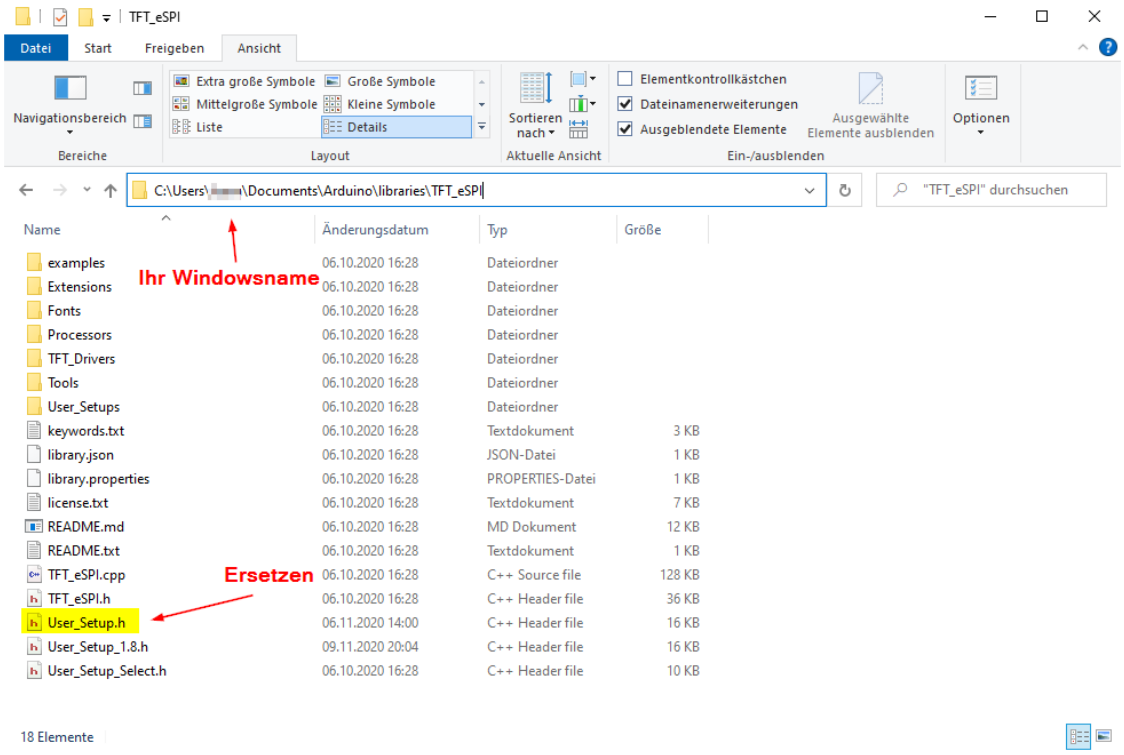

Abbildung 10: User_Setup.h überschreiben

Bei den neuen Az-Touch Mods ist es egal ob Sie die 2.4“- oder 2.8“-Variante benutzten. Da beide die gleiche Auflösung haben, kann der Quellcode ohne weitere Anpassungen direkt für beide Varianten genutzt werden.

Was bleibt nun noch zu tun? Nun wie oben beschrieben müssen Sie die Daten für WLAN und openweathermap.org mit Ihren Daten überschreiben. Des Weiteren können Sie die Farben nach Belieben verändern oder aber nach der Anleitung weiter oben, eigene Piktogramme für die Uhr verwenden. Für diejenigen unter Ihnen, die Spaß an Mathematik haben, Sie können den Zeiger andere Formen verpassen, der Kreativität sind keine Grenzen gesetzt. Da es noch ein bisschen Platz auf dem Display gibt, wäre sogar vorstellbar das Datum und den Wochentag in der Uhr anzuzeigen. Diese Informationen sind im Quellcode schon vorhanden, müssen nur noch an die richtige Position im Display visualisiert werden.

Ich hoffe Sie werden viel Freude mit der Uhr haben und das Interesse für einfache oder komplexere IoT-Projekte ist geweckt.

Weitere Projekte für Az-Delivery von mir, finden Sie unter https://github.com/M3taKn1ght/Blog-Repo.