PRÜFUNGSSTUDIENARBEIT

Angewandte Mikrocomputertechnik WS 2023/2024



https://www.alamy.de/fotos-bilder/bank-tresor.html?sortBy=relevant

Realisierung eines Codeschlosses

|  |  |
| --- | --- |
| vorgelegt von: | Michael Hiegl/22109460 |
| Prüfer: | Prof. Dr.-Ing. Andreas Penningsfeld |
| Modul: | MDP-24 FWP-Modul |

Inhaltsverzeichnis

[3 Einleitung 3](#_Toc1)

[4 Aufgabenstellung: 3](#_Toc2)

[5 Anforderungen: 3](#_Toc3)

[6 Ausarbeitung der Software – Master ESP 32: 4](#_Toc4)

[6.1 Übersicht über verwendeten Libraries 5](#_Toc5)

[6.2 Übersicht zu den Funktionen 7](#_Toc6)

[7 Ausarbeitung der Software – Slave ESP 8266 16](#_Toc7)

[7.1 Übersicht über verwendeten Libraries 16](#_Toc8)

[7.2 Übersicht zu den Funktionen 16](#_Toc9)

[7.3 Programm Master ESP32 (mit Überblick) 18](#_Toc10)

[7.4 Programm Slave ESP8266 (mit Überblick) 18](#_Toc11)

# Einleitung

Im Rahmen einer Projektarbeit soll eine vorgegebene Aufgabenstellung mit Hilfe eines Mikrocontrollers umgesetzt werden. Als Hardware stehen zwei Boards mit verschiedenen Peripherien (Sensoren, TFT-Touchscreen, Relais), ausgestattet mit einem ESP32 bzw. einem ESP8266 als Mikrocontroller.

# Aufgabenstellung:

Der Versuchsaufbau besteht aus dem 'ESP32-Board THD' und dem 'Remote Relais Modul'.

Es soll ein Code-Schloss (Tresor) realisiert werden.

Ein vierstelliger Code soll zum Schließen des Tresors einprogrammiert werden können und dient dann auch wieder zum Öffnen des Tresors. Das Schloss ist das Relais auf der Platine 'Remote Relais Modul'.

Der Code soll abgespeichert werden, so dass er auch nach einem Ausfall der Betriebsspannung noch zur Verfügung steht.

Unabhängig von diesem Code öffnet der sogenannte "Hausmeistercode" das Schloß immer.

Der Code ist**: *09913615516***

Die Benutzerführung und Anzeige des Codes sollen auf dem TFT-Display erfolgen.

Die Eingaben sollen entweder

1. über das Touch-Panel im TFT

**oder**

1. die Zehnertastatur …erfolgen.

Zusätzlich soll der Potiwert vom'Remote Relais Modul' auf dem TFT grafisch angezeigt werden.

Die Verbindung zwischen den beider Platinen erfolgt über WLAN.

* *Entscheidung für die* ***Option b)*** *!*

# Anforderungen:

Die einzelnen Punkte aus der Aufgabenstellung werden nun näher betrachtet und weiter eingegrenzt. Es lassen sich daraus auch weitere Anforderungen ableiten:

1. **Es soll ein vierstelliger Code einprogrammiert werden können.**

* Die Eingabemöglichkeit für den Code muss auf 4 Stellen begrenzt sein und mindestens 4 Stellen aufweisen.
* Der Code muss veränderbar sein nach Freischaltung durch den einprogrammierten bzw. dem letzten gesetzten Code.
* Der Code muss permanent gespeichert werden können.

1. **Über diesen Code kann dann das Relais auf der Remote Relais Platine geschaltet werden.**

* Da es sich um ein Codeschloss handelt ist eine gesicherte drahtlose Verbindung zwischen beiden Platinen erforderlich. Zur Umsetzung wird das auf Ethernet basierende, verbindungslose System ***ESP-NOW*** verwendet.
* Eine Möglichkeit zum Schalten des Relais und eine Anzeige über den Zustand des gelben Potis wird benötigt.

1. **Die Benutzerführung und Anzeige des Codes sollen auf dem TFT-Display erfolgen.**

* Da auf dem TFT der Treiber ILI934 verbaut und dieser geläufig ist, kann zur Darstellung auf diesem auf eine bestehenden Libraries aus dem Skript zurückgegriffen werden.

1. **Die Eingaben über die Zehnertastatur.**

Die Eingabe über die Matrixtastatur welche über I2C an den ESP32 angeschlossen ist.

Die Library dazu wurde aus einem Internetforum entnommen welches diese sehr empfohlen hat.

1. **Das Codeschloss soll über einen Hausmeistercode verfügen**

* Das Codeschloss soll sich mithilfe eines Mastercodes zurücksetzen lassen.
* Die Länge des Mastercode soll dabei über mehr als 4 Stellen verfügen. Es wurde hierfür eine Handynummer verwendet.

# Ausarbeitung der Software – Master ESP 32:

Da der Einsatzort des Codeschlosses nicht genauer festgelegt ist, wird bei der Ausgestaltung ein besonderer Wert auf eine vereinfachte Bedienung gelegt, deshalb erscheint bei einschalten des Boards eine kurze Anleitung zu den Features.

**Erstinbetriebnahme**

Bei erstmaliger Bedienung muss das Board bzw. der EEPROM speziel beschrieben werden damit ein Wert in ihm steht mit dem verglichen werden kann, dies erfolgt durch das einmalige Flashen eines leicht abgeänderten Quellcodes, anschließend kann der originale Quellcode auf das Board geschrieben werden wobei der abgeänderte Code auskommentiert/gelöscht werden muss.

Um nun einen Code einzugeben muss als erstes der Hausmeistercode  ***09913615516*** eingegeben werden, dann gelangt man in den berechtigten Modus von dem aus man über ein Untermenu einen neuen Benutzerdefinierten Code einprogrammieren kann.

**Standardbedienung**

Von nun an kann sowohl mit dem Hausmeistercode als auch mit dem Benutzercode in den ***berechtigen Modus*** gewechselt werden.

* *Das aufleuchten der Grünen LED zeigt an dass das Board erfolgreich als ESP-NOW Gerät initialisiert ist.*

Bei der Eingabe eines richtigen Codes erscheint im TFT Bildschirm ein Untermenü welches 2 Optionen enthält:

Option Taste **A**: Zurückwechseln zu unberechtigter Modus (Lockout-funktion)

Optionb Taste **C**: Neuen Code vergeben (4-Stellig)

Im berechtigten Modus kann mit dem Master-Poti(Rot) welches als „Schlüssel“ Funktioniert Auf- oder Zugesperrt werden, hierbei wird eine Nachricht von einem Char an den ESP8266 über ESP-NOW gesendet welche die Relais Platine auswertet und je nach Nachricht Aus- oder Ein-schaltet.

Bei der Wahl von Option C kann ein neuer Code gewählt und anschließend mit B bestätigt werden.

Da der Code allgemein auch nach einem Spannungsverlust erhalten bleiben soll wird er in das EEPROM des ESP32 geschrieben.

Der Schaltzustand der zweiten Platine (Relais) kann bei entsperrtem Schloss übers Poti geändert werden, das Poti ist daher als „Schlüssel“ zu sehen bei dem das Drehen im Uhrzeigersinn zum „Aufsperren“ und im Gegenuhrzeigersinn zum „Zusperren“ führt.

Grundsätzlich kann in jedem Modus mit tippen auf den**Digitast** im ESP32 Boardder Wert des gelben Poti ausgelesen werden, hierzu erscheint ein Drehzeigerinstrument im Display welches visuell den aktuellen Wert darstellt.

Die gesamte Software wurde in der Arduino IDE erstellt. Diese IDE ermöglicht gleichzeitig auch das Flashen des Microcontrollers und ein einfaches Management für benötigte Librarys und Toolchains für unterschiedliche Boards.

## Übersicht über verwendeten Libraries

1***. WiFi.h*** und ***esp\_wifi.h***: Diese Bibliothek ermöglicht die Verbindung zu WLAN-Netzwerken und wird auf ESP32-Mikrocontrollern verwendet. Mit dieser Bibliothek können Sie den ESP32 in den Station- oder Access Point-Modus versetzen um als WLAN-Client oder als WLAN-Access Point zu agieren in unserem Fall sind das ESP32 und das ESP 8266 beides zugleich, weiter Aufgabe ist Netzwerke zu suchen und sich mit ihnen zu verbinden in unserem Fall wird sie benutzt um den beiden Boards neue MAC-Adressen zuzuweisen

2***. esp\_now.h***: Diese Bibliothek ermöglicht die Kommunikation über ESP-NOW, einem protokollübergreifenden Kommunikationsprotokoll, das für den schnellen und energieeffizienten Datenaustausch zwischen Wifi-fähigen ESP-Plattformen entwickelt wurde.

*ESP-NOW im Überblick:*

ESP-NOW ist ein drahtloses Kommunikationsprotokoll, das speziell für die Mikrocontroller ESP8266 und ESP32 entwickelt wurde. Es ermöglicht eine direkte Punkt-zu-Punkt-Kommunikation ohne die Notwendigkeit eines zentralen WLAN-Routers. Dies reduziert Netzwerkkonfiguration und -komplexität.

Die Vorteile von ESP-NOW liegen in der geringen Energieanforderung, schnellen Datenübertragung und einfachen Einrichtung. Die Peer-to-Peer-Kommunikation ermöglicht es, dass jedes Gerät als Sender oder Empfänger fungieren kann. Vor der Verwendung müssen die Geräte ihre MAC-Adressen vorab konfigurieren.

3. ***SPI.h:*** Die SPI-Bibliothek bietet Funktionen für die serielle Peripherieschnittstelle (Serial Peripheral Interface). SPI wird häufig für die Kommunikation zwischen Mikrocontrollern und Peripheriegeräten wie Sensoren, Displays und Speicherchips verwendet.

4***. TFT\_eSPI***: Diese Bibliothek ermöglicht die Ansteuerung von Farb-TFT-Displays, insbesondere für ESP8266 und ESP32. Sie bietet Funktionen zum Zeichnen von Text, Grafiken und Formen auf dem Display.

Anmerkung: beim Arbeiten mit dieser Library wurden wir auf ein Probleme aufmerksam welches durch ein Update ausgelöst wurde und einiges an Zeit und Nerven gekostet hat, der Bildschirm blieb nach dem Flashen Weiß und gab nichts mehr aus, behoben wurde dies endgültig durch das Benutzen und neu Aufspielen der I-Learn Bibliothek Version. Der Grund für den Fehler ist das fehlen des korrekten Inhaltes der ***User\_Setup\_Select.h*** nach einem Update.

Betreiber der Library Bodmer schreibt dazu in seinem Blog:

*„Die Aktualisierungen für den ESP32 S2/C3/S3 bedeuten, dass die Bibliothek das ESP32 Arduino Board Package in der Version 2.x.x oder neuer erfordert.*

*Der Bildschirmcontroller, die Schnittstellenpins und die Konfigurationseinstellungen der Bibliothek müssen innerhalb der Bibliothek definiert werden. Sie dürfen NICHT im Arduino-Sketch definiert werden. Details dazu finden Sie in der Datei User\_Setup\_Select.h. Dieser Ansatz hat erhebliche Vorteile, da er die Beispiele von langen Konfigurationsoptionen frei hält und einmal definiert, kann jedes Beispiel ohne Änderungen ausgeführt werden. Nutzer von PlatformIO können diese Einstellungen projektweise in einer platformio.ini-Datei festlegen. Weitere Informationen dazu finden Sie im "Docs"-Ordner in der Bibliothek.“*

5. ***Key.h***: Diese Bibliothek ist Teil des Arduino Keypad-Systems und wird verwendet, um Tasten auf einem physischen Tastenfeld zu verwalten und abzufragen. Sie bietet Funktionen zur Abfrage von Tastenstatus und Ereignissen.

6***. Keypad.h***: Diese Bibliothek erweitert die Funktionalität von Key.h und bietet erweiterte Funktionen zur Handhabung von Tastatureingaben. In Ihrem Code wird eine abgeleitete Klasse namens Keypad\_I2C verwendet, die speziell für den Einsatz mit I2C-Tastenfeldern entwickelt wurde.

7. ***Keypad\_I2C.h***: Diese Bibliothek ist eine Erweiterung der Keypad-Bibliothek und ermöglicht die Verwendung von I2C zur Kommunikation mit dem Tastenfeld.

*8.* ***EEPROM.h***: Mit dieser Bibliothek können Sie den EEPROM (Electrically Erasable Programmable Read-Only Memory) auf dem Mikrocontroller verwenden, um Daten dauerhaft zu speichern. In Ihrem Code wird es verwendet, um den Code für den Zugriff auf bestimmte Funktionen zu speichern und beim Neustart beizubehalten.

## Übersicht zu den Funktionen

Die einzelnen Funktionen des Programms wurden der Übersicht halber logisch gruppiert. Außerdem wird versucht auf die verwendeten Methoden, Funktionen und Libraries einzugehen und diese kurz zu beschreiben.

**Includes und Defines**

#define I2CADDR 0x20  // Die I2C-Adresse für die Tastatur

#define M\_SIZE 1.3333  // Größenfaktor für das Messgerät

#define TFT\_GREY 0x5AEB  // Grauton für das TFT-Display

#define EEPROM\_SIZE 20  // Größe des EEPROM-Speicherbereichs in Bytes

// Pin-Definitionen

#define LED\_GREEN 2    // Pin für die grüne LED

#define TASTLED 26      // Pin für die Tasten-LED

#define POTI 34         // Pin für das Potentiometer

#define TASTER 39       // Pin für den Taster

// Verwendete Bibliotheken

#include <WiFi.h>          // WiFi-Bibliothek für die drahtlose Kommunikation

#include <esp\_wifi.h>      // ESP32 WiFi-Bibliothek für erweiterte Funktionen

#include <esp\_now.h>       // ESP-NOW-Bibliothek für die drahtlose Kommunikation zwischen ESP-Boards

#include <SPI.h>           // SPI-Bibliothek für die Kommunikation mit Peripheriegeräten über das SPI-Protokoll

#include <TFT\_eSPI.h>      // TFT\_eSPI-Bibliothek für die Steuerung von TFT-Displays

#include <Key.h>           // Bibliothek für die Handhabung von Tasten

#include <Keypad.h>        // Bibliothek für die Verwendung von Tastatur-ähnlichen Eingabegeräten

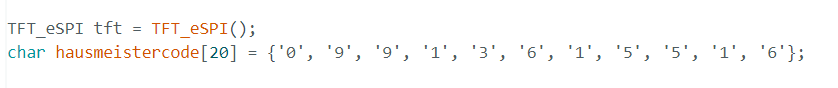
#include <Keypad\_I2C.h>    // I2C-Erweiterung für die Keypad-Bibliothek

#include <EEPROM.h>        // Bibliothek für den Zugriff auf den EEPROM-Speicher

Hier werden einige Makros für I2C-Adresse, Größenfaktor, Grauton, EEPROM-Größe und Pin-Definitionen festgelegt. Außerdem werden die benötigten Bibliotheken eingebunden.

**TFT und Hausmeistercode**

Die Zeile TFT\_eSPI tft = TFT\_eSPI(); erstellt ein Objekt der Klasse TFT\_eSPI mit dem Namen tft, um auf das TFT-Display zuzugreifen. Ein vordefinierter Hausmeistercode wird festgelegt.



**Meter- und Tastenfeld-Konfiguration**

float ltx = 0;

uint16\_t osx = M\_SIZE \* 120, osy = M\_SIZE \* 120;

int old\_analog = -999;

int value[6] = {0, 0, 0, 0, 0, 0};

int old\_value[6] = {-1, -1, -1, -1, -1, -1};

int d = 0;

Verschiedene Variablen werden für die Anzeige eines analogen Meters und die Überwachung von Tasten im Tastenfeld definiert damit später darauf zugegriffen werden kann.

**Tastenfeld-Konfiguration**

#const byte ROWS = 4;

const byte COLS = 4;

char keys[ROWS][COLS] = {

  {'1', '4', '7', '\*'},

  {'2', '5', '8', '0'},

  {'3', '6', '9', '#'},

  {'A', 'B', 'C', 'D'}

};

byte rowPins[ROWS] = {0, 1, 2, 3};

byte colPins[COLS] = {4, 5, 6, 7};

Keypad\_I2C keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS, I2CADDR, PCF8574);

Die Konfiguration für das Tastenfeld (Keypad) wird erstellt, einschließlich der Tastenbelegung und der Pin-Konfiguration.

**Für Benutzeroberfläche**

int mode = 0;

char inputBuffer[20];

int inputIndex = 0;

char validCode[20] = {'A'};

Variablen für den aktuellen Modus, einen Eingabepuffer, den Pufferindex und einen gültigen Code werden definiert.

0= Unberechtigt

1= Berechtigt

2= Neue Code vergabe

**ESP-Now**

const uint8\_t newMacAddress[] = {0x94, 0x3C, 0xC6, 0x33, 0x68, 0x01};

const uint8\_t receiverAddress[] = {0x96, 0x3B, 0xC7, 0x34, 0x69, 0x02};

esp\_now\_peer\_info\_t peerInfo;

int potiwert;

char message\_on[] = "E";

char message\_off[] = "A";

Konfigurationen für die ESP-NOW Kommunikation werden festgelegt, einschließlich MAC-Adressen, Peer-Informationen, Potentiometer-Wert und Nachrichten

**Funktion zum Skalieren des empfangenen Potiwerts**

void readpoti(const uint8\_t\* macAddr, const uint8\_t\* incomingData, int len) {

  potiwert = int(\*incomingData / 2.04);

}

Diese Funktion wird aufgerufen, wenn ein Wert über ESP-NOW empfangen wird. Sie skaliert den empfangenen Wert auf einen Wert zwischen 0 und 100 und speichert ihn in der Variable ***potiwert***

**Funktion zum Versenden von Ein-/Ausschaltnachrichten**

void lock() {

  if (analogRead(POTI) <= 2040) {

    esp\_now\_send(receiverAddress, (uint8\_t\*)message\_off, sizeof(message\_off));

  }

  if (analogRead(POTI) > 2050) {

    esp\_now\_send(receiverAddress, (uint8\_t\*)message\_on, sizeof(message\_on));

  }

}

Diese Funktion überprüft den Roten Potentiometerwert und sendet über ESP-NOW Nachrichten zum Ein- oder Ausschalten und sendet je nach Poti wert eine aus oder Einschaltnachricht, diese Funktion wird nur im Berechtigten Modus ausgeführt deshalb kann nur in diesem Modus auf und zugesperrt werden.

**Funktion zum Löschen des InputBuffers**

void clearbuffer() {

  inputIndex = 0;

  inputBuffer[inputIndex] = '\0';

  inputIndex = 0; }

Diese Funktion setzt den Eingabepuffer zurück

**Hauptfunktion setup() (nur bei Neustart des Boards)**

void setup() {

  // Initialisierung von Wire für die I2C-Kommunikation

  Wire.begin();

  // Initialisierung des Keypads

  keypad.begin(makeKeymap(keys));

  // Konfiguration der Pin-Modi

  pinMode(LED\_GREEN, OUTPUT);

  pinMode(TASTER, INPUT);

  // ESP32 in den Station-Modus versetzen und neue MAC-Adresse zuweisen

  WiFi.mode(WIFI\_STA);

  esp\_wifi\_set\_mac(WIFI\_IF\_STA, newMacAddress);

  // Initialisierung von ESPnow

  if (esp\_now\_init() != ESP\_OK) {

    digitalWrite(LED\_GREEN, LOW);

  } else {

    digitalWrite(LED\_GREEN, HIGH);

  }

  // Kopieren der ESP8266-MAC-Adresse in peerInfo

  memcpy(peerInfo.peer\_addr, receiverAddress, 6);

  peerInfo.channel = 0;

  peerInfo.encrypt = false;

  // Hinzufügen des ESP8266 als Kommunikationspartner

  esp\_now\_add\_peer(&peerInfo);

  // Registrieren der Funktion zum Empfangen von Daten über ESPnow

  esp\_now\_register\_recv\_cb(readpoti);

  // Beginnen der EEPROM-Nutzung

  EEPROM.begin(EEPROM\_SIZE);

  // Laden des im EEPROM gespeicherten Codes

  for (int i = 0; i < 20; i++) {

    validCode[i] = EEPROM.read(i);

  }

   // validCode[0] = {'A'}; ist auskommentiert, wird nur bei der Erstinitialisierung gebraucht um einen Wert in den EEPROM zu schreiben

  // TFT-Initialisierung

  tft.init();

  tft.setRotation(1);

  tft.fillScreen(TFT\_BLACK);

}

In dieser Funktion werden alle notwendigen Initialisierungen durchgeführt, einschließlich der Konfiguration des Keypads, der WiFi- und ESP-NOW-Einstellungen, der EEPROM-Nutzung, der TFT-Initialisierung und anderer erforderlicher Schritte.

**Hauptfunktion loop()**

void loop() {

  // Überprüfung, ob der Taster nicht gedrückt ist

  if (digitalRead(TASTER) == HIGH) {

    char rawkey = keypad.getKey();

    char key;

    // Filtern von A, B, C aus dem Keypad-Input

    if (rawkey != 'A' && rawkey != 'B' && rawkey != 'C') {

      key = rawkey;

    }

    // Überprüfung, ob ein Tastendruck vorliegt (außer A, B, C)

    if (key != NO\_KEY) {

      if (key == 'D') {

        // Löschen des InputBuffers bei D

        clearbuffer();

      } else {

        // Anhängen der Taste an den InputBuffer

        inputBuffer[inputIndex++] = key;

        inputBuffer[inputIndex] = '\0';

      }

    }

    // Überprüfung, ob der InputBuffer dem Hausmeistercode oder gültigem Code entspricht

    if (String(inputBuffer) == String(hausmeistercode) || String(inputBuffer) == String(validCode)) {

      mode = 1;

      clearbuffer();

    }

      if (rawkey == 'C' && mode == 1) {  // Wenn im berechtigten Modus und Wunsch auf neuen Code geäußert

      mode = 2;                        // In Codeänderungsmodus wechseln

      clearbuffer();

    }

    // Überprüfung, ob im berechtigten Modus und A gedrückt

    if (rawkey == 'A' && mode == 1) {

      clearbuffer();

      mode = 0;

    }

    // Überprüfung, ob im Codeänderungsmodus und B gedrückt

    if (mode == 2 && rawkey == 'B') {

      for (int i = 0; i < 20; i++) {

        validCode[i] = inputBuffer[i];

      }

      mode = 1;

    }

    // Sicheres Nullterminieren von validCode und InputBuffer

    inputBuffer[19] = '\0';

    validCode[19] = '\0';

    // TFT-Hintergrund zurücksetzen und Formatierung einstellen

    tft.fillScreen(TFT\_BLACK);

    tft.setCursor(0, 0, 2);

    tft.setTextColor(TFT\_GREEN, TFT\_BLACK);

    tft.setTextSize(1);

    // Berechtigter Modus

    if (mode == 1) {

      tft.println("Berechtigter Modus");

      tft.println("Mit C neuen Code eingeben");

      tft.println("Mit A verlassen");

      lock();

    }

    // Codeänderungsmodus

    else if (mode == 2) {

      tft.println("Neuen Code eingeben:");

      tft.println(String(inputBuffer));

      tft.println("Mit B bestaetigen");

      tft.println("Nur Ziffern erlaubt");

      tft.println("Neuer Code darf nicht alter Code sein");

      esp\_now\_send(receiverAddress, (uint8\_t\*)message\_off, sizeof(message\_off));

    }

    // Unberechtigter Modus

    else {

      tft.println("Unberechtigter Modus");

      tft.println("Bitte Code eingeben:");

      tft.println(String(inputBuffer));

      tft.println("Mit D kann immer geloescht werden");

      tft.println(" ");

      tft.println("Mit tippen auf den Taster kann der Potiwert");

      tft.println("angezeigt werden");

      esp\_now\_send(receiverAddress, (uint8\_t\*)message\_off, sizeof(message\_off));

    }

    delay(100);

  } else {

    analogMeter();

    plotNeedle(potiwert, 0);

    delay(2000);

  }

  // Aktualisieren der EEPROM-Werte

  for (int i = 0; i < 20; i++) {

    EEPROM.write(i, validCode[i]);

  }

  EEPROM.commit();

}

Die **loop()**-Funktion enthält den Hauptcode, der in einer Schleife ausgeführt wird. Hier wird das Tastenfeld überwacht, die Berechtigungsmodi überprüft, das Meter aktualisiert, die Nadel gezeichnet und die EEPROM-Werte aktualisiert.

**Funktion zum Zeichnen des Meters auf dem TFT-Display**

// Meter-Funktion zum Zeichnen des Meters auf dem TFT-Display

void analogMeter() {

  // Meter-Umrandung zeichnen

  tft.fillRect(0, 0, M\_SIZE \* 239, M\_SIZE \* 126, TFT\_GREY);

  tft.fillRect(5, 3, M\_SIZE \* 230, M\_SIZE \* 119, TFT\_WHITE);

  // Textfarbe festlegen

  tft.setTextColor(TFT\_BLACK);

  // Ticks alle 5 Grad von -50 bis +50 Grad zeichnen

  for (int i = -50; i < 51; i += 5) {

    // Lange Skalentick-Länge

    int tl = 15;

    // Koordinaten des Ticks berechnen

    float sx = cos((i - 90) \* 0.0174532925);

    float sy = sin((i - 90) \* 0.0174532925);

    uint16\_t x0 = sx \* (M\_SIZE \* 100 + tl) + M\_SIZE \* 120;

    uint16\_t y0 = sy \* (M\_SIZE \* 100 + tl) + M\_SIZE \* 140;

    uint16\_t x1 = sx \* M\_SIZE \* 100 + M\_SIZE \* 120;

    uint16\_t y1 = sy \* M\_SIZE \* 100 + M\_SIZE \* 140;

    // Koordinaten des nächsten Ticks für Zonenfüllung berechnen

    float sx2 = cos((i + 5 - 90) \* 0.0174532925);

    float sy2 = sin((i + 5 - 90) \* 0.0174532925);

    int x2 = sx2 \* (M\_SIZE \* 100 + tl) + M\_SIZE \* 120;

    int y2 = sy2 \* (M\_SIZE \* 100 + tl) + M\_SIZE \* 140;

    int x3 = sx2 \* M\_SIZE \* 100 + M\_SIZE \* 120;

    int y3 = sy2 \* M\_SIZE \* 100 + M\_SIZE \* 140;

    // Grüne Zone für Werte von 0 bis 25 Grad

    if (i >= 0 && i < 25) {

      tft.fillTriangle(x0, y0, x1, y1, x2, y2, TFT\_GREEN);

      tft.fillTriangle(x1, y1, x2, y2, x3, y3, TFT\_GREEN);

    }

    // Orangene Zone für Werte von 25 bis 50 Grad

    if (i >= 25 && i < 50) {

      tft.fillTriangle(x0, y0, x1, y1, x2, y2, TFT\_ORANGE);

      tft.fillTriangle(x1, y1, x2, y2, x3, y3, TFT\_ORANGE);

    }

    // Kurze Skalentick-Länge

    if (i % 25 != 0) tl = 8;

    // Ticks zeichnen

    x0 = sx \* (M\_SIZE \* 100 + tl) + M\_SIZE \* 120;

    y0 = sy \* (M\_SIZE \* 100 + tl) + M\_SIZE \* 140;

    x1 = sx \* M\_SIZE \* 100 + M\_SIZE \* 120;

    y1 = sy \* M\_SIZE \* 100 + M\_SIZE \* 140;

    tft.drawLine(x0, y0, x1, y1, TFT\_BLACK);

    // Label zeichnen, wenn erforderlich

    if (i % 25 == 0) {

      x0 = sx \* (M\_SIZE \* 100 + tl + 10) + M\_SIZE \* 120;

      y0 = sy \* (M\_SIZE \* 100 + tl + 10) + M\_SIZE \* 140;

      switch (i / 25) {

        case -2: tft.drawCentreString("0", x0, y0 - 12, 2); break;

        case -1: tft.drawCentreString("25", x0, y0 - 9, 2); break;

        case 0: tft.drawCentreString("50", x0, y0 - 7, 2); break;

        case 1: tft.drawCentreString("75", x0, y0 - 9, 2); break;

        case 2: tft.drawCentreString("100", x0, y0 - 12, 2); break;

      }

    }

    // Bogen der Skala zeichnen

    sx = cos((i + 5 - 90) \* 0.0174532925);

    sy = sin((i + 5 - 90) \* 0.0174532925);

    x0 = sx \* M\_SIZE \* 100 + M\_SIZE \* 120;

    y0 = sy \* M\_SIZE \* 100 + M\_SIZE \* 140;

    if (i < 50) tft.drawLine(x0, y0, x1, y1, TFT\_BLACK);

  }

  // Bezellinie zeichnen

  tft.drawRect(5, 3, M\_SIZE \* 230, M\_SIZE \* 119, TFT\_BLACK);

  // Nadel auf 0 setzen

  plotNeedle(0, 0);

}

Diese Funktion zeichnet das analoge Meter auf dem TFT-Display. Sie enthält Konfigurationen für verschiedene Farbzonen und Skalierungsticken.

**Funktion zum Zeichnen der Nadel abhängig vom Wert**

// Funktion zum Zeichnen der Nadel abhängig vom Wert

void plotNeedle(int value, byte ms\_delay) {

  // Textfarbe festlegen

  tft.setTextColor(TFT\_BLACK, TFT\_WHITE);

  // Wert als Zeichenkette konvertieren

  char buf[8];

  dtostrf(value, 4, 0, buf);

  // Wert rechtsbündig am oberen Rand des Displays anzeigen

  tft.drawRightString(buf, M\_SIZE \* 40, M\_SIZE \* (119 - 20), 2);

  // Wertbegrenzung, um Endanschläge der Nadel zu emulieren

  if (value < -10) value = -10;

  if (value > 110) value = 110;

  // Nadelbewegung, bis neuer Wert erreicht ist

  while (!(value == old\_analog)) {

    // Nadelposition für den neuen Wert berechnen

    if (old\_analog < value) old\_analog++;

    else old\_analog--;

    // Sofort aktualisieren, wenn die Verzögerung 0 ist

    if (ms\_delay == 0) old\_analog = value;

    // Winkel der Nadel in Grad berechnen

    float sdeg = map(old\_analog, -10, 110, -150, -30);

    // Koordinaten der Nadel-Spitze berechnen

    float sx = cos(sdeg \* 0.0174532925);

    float sy = sin(sdeg \* 0.0174532925);

    // X-Delta des Nadelanfangs berechnen (beginnt nicht am Drehpunkt)

    float tx = tan((sdeg + 90) \* 0.0174532925);

    // Altes Nadelbild löschen

    tft.drawLine(M\_SIZE \* (120 + 20 \* ltx - 1), M\_SIZE \* (140 - 20), osx - 1, osy, TFT\_WHITE);

    tft.drawLine(M\_SIZE \* (120 + 20 \* ltx), M\_SIZE \* (140 - 20), osx, osy, TFT\_WHITE);

    tft.drawLine(M\_SIZE \* (120 + 20 \* ltx + 1), M\_SIZE \* (140 - 20), osx + 1, osy, TFT\_WHITE);

    // Text unter der Nadel neu zeichnen

    tft.setTextColor(TFT\_BLACK);

    // Neue Koordinaten der Nadel-Spitze für das nächste Löschen speichern

    ltx = tx;

    osx = M\_SIZE \* (sx \* 98 + 120);

    osy = M\_SIZE \* (sy \* 98 + 140);

    // Nadel an neuer Position zeichnen, Magenta macht die Nadel etwas kräftiger

    tft.drawLine(M\_SIZE \* (120 + 20 \* ltx - 1), M\_SIZE \* (140 - 20), osx - 1, osy, TFT\_RED);

    tft.drawLine(M\_SIZE \* (120 + 20 \* ltx), M\_SIZE \* (140 - 20), osx, osy, TFT\_MAGENTA);

    tft.drawLine(M\_SIZE \* (120 + 20 \* ltx + 1), M\_SIZE \* (140 - 20), osx + 1, osy, TFT\_RED);

    // Nadel leicht verlangsamen, wenn sie sich der neuen Position nähert

    if (abs(old\_analog - value) < 10) ms\_delay += ms\_delay / 5;

    // Warten vor der nächsten Aktualisierung

    delay(ms\_delay);

  }

}

Diese Funktion zeichnet die Nadel des Meters abhängig vom übergebenen Wert. Der Wert wird allmählich animiert, und die Verzögerung zwischen den Animationsschritten kann eingestellt werden.

# **Ausarbeitung der** Software – Slave ESP 8266

Das Programm des Slave ist im Vergleich zum Master im Umfang deutlich kompakter. Verschiedene Funktionen sind dabei identisch zum Master, speziell die drahtlose Verbindung betreffend.

## Übersicht über verwendeten Libraries

**ESP8266WiFi.h**: Diese Bibliothek ist eine Schlüsselbibliothek für die WiFi-Kommunikation auf ESP8266-Mikrocontrollern. Der ESP8266 ist ein kostengünstiger und leistungsstarker Mikrocontroller mit integriertem WiFi-Modul, der oft in IoT-(Internet of Things)-Projekten verwendet wird. Die Bibliothek bietet Funktionen zur Konfiguration und Verwaltung von WiFi-Verbindungen auf dem ESP8266.

**espnow**.: Diese Bibliothek ist spezifisch für die ESP-NOW-Kommunikation auf ESP8266-Mikrocontrollern. ESP-NOW ermöglicht die direkte Kommunikation zwischen ESP8266-Geräten ohne Verbindung zu einem herkömmlichen WiFi-Netzwerk. Dies ist besonders nützlich für die Kommunikation zwischen verschiedenen ESP8266-Modulen in einem lokalen Netzwerk.

## Übersicht zu den Funktionen

**Definitionen und Bibliotheken**

#define RELAISPIN 16

#define POTI A0

#include <ESP8266WiFi.h>

#include <espnow.h>

Der Code verwendet die <ESP8266WiFi.h>-Bibliothek für die WiFi-Konfiguration und die <espnow.h>-Bibliothek für die ESP-NOW-Kommunikation.

Es werden einige Konstanten definiert, darunter der Pin für das Relais (RELAISPIN) und der Analogpin für das Potentiometer (POTI).

**MAC-Adressen**

uint8\_t newMacAddress[] = { 0x96, 0x3B, 0xC7, 0x34, 0x69, 0x02 };

uint8\_t receiverAddress[] = { 0x94, 0x3C, 0xC6, 0x33, 0x68, 0x01 };

Hier werden die MAC-Adressen für den Sender (newMacAddress) und den Empfänger (receiverAddress) definiert.

**Funktionen**

//empfängt die MAC-Adresse

void onoff(uint8\_t\* macAddr, uint8\_t\* incomingData, uint8\_t len) {

  // Nimmt empfangene Nachricht als Argument schaltet je nach Inhalt das Relais HIGH oder LOW

  if (incomingData[0] == 'E') {

    digitalWrite(RELAISPIN, HIGH);

  }

  if (incomingData[0] == 'A') {

    digitalWrite(RELAISPIN, LOW);

  }

}

void peercheck(uint8\_t\* mac\_addr, uint8\_t sendStatus) {   // Überprüft, ob der Teilnehmer erreicht werden konnte. Schaltet das Relais ab, wenn dies nicht der Fall ist

  if (sendStatus != 0) {

    digitalWrite(RELAISPIN, LOW); //Sicherheitsfunktion

  }

}

***onoff***: Diese Funktion wird aufgerufen, wenn eine Nachricht empfangen wird. Sie schaltet das Relais ein (HIGH), wenn die empfangene Nachricht "E" ist, und schaltet aus(LOW), wenn die Nachricht "A" ist.

***peercheck:*** Diese Funktion wird aufgerufen, wenn eine Nachricht gesendet wird. Sie überprüft den Sendestatus, und wenn die Übertragung nicht erfolgreich war, schaltet sie das Relais aus (LOW).

Interruptfunktion die aufgreufen wird wenn ESP8266 eine Nachricht sendet

Überprüft send status ob 0 und wenn nicht 0 -> abschalten des Relais

Sicherheitsgrund: bei fehlenden Kommunikationspartner wird zugesperrt.

**Setup-Funktion:**

void setup() {

  pinMode(RELAISPIN, OUTPUT);         // Definieren von Relaispin als Output

  pinMode(LED\_BUILTIN, OUTPUT);       // Definieren von eingebauter LED als Output

  digitalWrite(RELAISPIN, LOW); // Zusperren im Fall eines Stromausfalls und Wiederanlaufs

  WiFi.mode(WIFI\_STA);               // Initialisieren des Boards als Wifi-Station

  wifi\_set\_macaddr(STATION\_IF, newMacAddress); // Vergeben einer MAC an das Board

  WiFi.disconnect(); // Deinitialisieren des Boards als Standard-Wifi-Device (nur ESP 8266)

  if (esp\_now\_init() != 0) {  // Initialisieren des Boards als ESPnow-Wifi-Device und Abfrage ob erfolgreich durch Ansteuern der eingebauten LED

    digitalWrite(LED\_BUILTIN, HIGH);

  } else {

    digitalWrite(LED\_BUILTIN, LOW);

  }

  esp\_now\_add\_peer(receiverAddress, ESP\_NOW\_ROLE\_COMBO, 0, NULL, 0);    // ESP32 als Kommunikationspartner hinzufügen. Dazu werden MAC, Rolle(Transceiver), Channel, Schlüssel(nicht existent Null) und Schlüssellänge übergeben

  esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_COMBO);   // ESP8266 zum Transciever machen

  esp\_now\_register\_recv\_cb(onoff);       // Führt bei Empfangen einer Nachricht die Interrupt-Funktion onoff aus. Definiert in ESPnow erhält diese als Argumente die Sender-MAC, die Nachricht und die Länge der Nachricht

esp\_now\_register\_send\_cb(peercheck);    // Führt bei Senden einer Nachricht die Interrupt-Funktion peercheck aus. Definiert in ESPnow erhält diese als Argumente die Empfänger-MAC, und den Erfolgsstatus des Versendens

}

Hier werden Pin-Modi gesetzt, die WiFi-Konfiguration durchgeführt, ESP-NOW initialisiert und die Peer-Adresse für die Kommunikation hinzugefügt.

Die Funktionen onoff und peercheck werden als Callback-Funktionen für den Empfang bzw. das Senden von Daten registriert.

**LOOP Funktion:**

void loop() {

  uint8\_t potiwert = analogRead(POTI) / 5;

// Einlesen des gelben Potiwertes an Analog Pin und runterskalieren des Wertes da nur 8 bit übertragbar und Analog Read leider ein Float ist welcher mehr als 8 bit hat deshalb umwandlung in int und teilung durch 5 im ESP 32 wird der wert dann wieder Hochskaliert

  esp\_now\_send(receiverAddress, (uint8\_t\*)&potiwert, sizeof(potiwert));

  // Versenden des Potiwertes

  delay(200);

}

//In der loop-Funktion wird der Wert des Potentiometers eingelesen, skaliert und über ESP-NOW alle 200ms an den Empfänger gesendet.

In der loop-Funktion wird der Wert des Potentiometers eingelesen, skaliert und über ESP-NOW an den Empfänger gesendet.

## Programm Master ESP32 (mit Überblick)

**Siehe** **Anhang A**

## Programm Slave ESP8266 (mit Überblick)

**Siehe Anhang B**