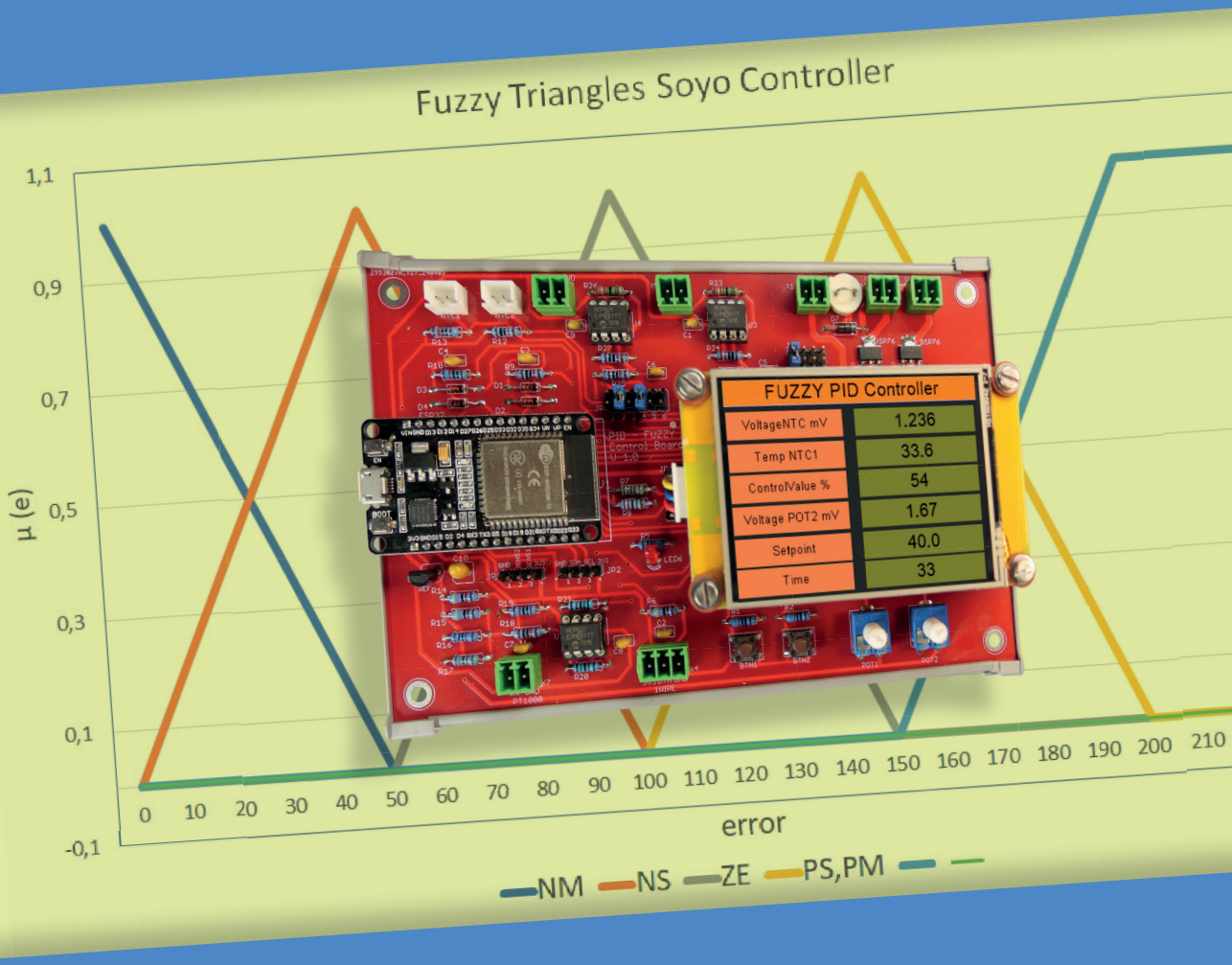
elektor books

Regelungstechnik mit Fuzzy Logic

Praktische Anwendungen und Projekte mit Arduino, ESP32 und RP2040



Josef Bernhardt

elektor
design > share > earn

Regelungstechnik mit Fuzzy Logic

Praktische Anwendungen und Projekte mit Arduino, ESP32 und RP2040

Josef Bernhardt

elektor
design › share › earn

1. Auflage 2024

ISBN 978-3-89576-646-6 Print
ISBN 978-3-89576-647-3 eBook

Satz und Aufmachung: D-Vision, Julian van den Berg | Oss (NL)
Druck: Ipskamp Printing, Enschede (NL)

Elektor Verlag GmbH, Aachen
www.elektor.de

Elektor ist die weltweit wichtigste Quelle für technische Informationen und Elektronik-Produkte für Maker, Ingenieure und Elektronik-Entwickler und für Firmen, die diese Fachleute beschäftigen. Das internationale Team von Elektor entwickelt Tag für Tag hochwertige Inhalte für Entwickler und DIY-Elektroniker, die über verschiedene Medien (Magazine, Videos, digitale Medien sowie Social Media) in zahlreichen Sprachen verbreitet werden. **www.elektor.de**

Inhaltsverzeichnis

Vorwort

Dieses Buch wurde für Studierende sowie für Elektroniker geschrieben, die sich für praktische Anwendungen der Regelungstechnik interessieren. Das Ziel dieses Buches ist, die Grundlagen der Regelungstechnik schwerpunktmäßig für Fuzzy Logic Regelungstechnik zu vermitteln und an mehreren praktischen Beispielen zu erläutern.

Insbesondere werden die Grundlagen der Sensoren und Linearisierung verständlich dargestellt, damit Sie ein solides Verständnis für die Erfassung von Sensordaten erhalten. Darüber hinaus bieten wir Ihnen detaillierte Informationen und Beispiele zur Ansteuerung verschiedener Aktuatoren, um Ihnen zu zeigen, wie Sie Ihre Projekte aufbauen können.

Das Buch wird durch zahlreiche Beispiele ergänzt und beschreibt reale Anwendungen der Regelungstechnik, die alle auf der Arduino-Plattform umgesetzt werden. Dies ermöglicht den Lesern, praktische Erfahrungen zu sammeln und das Gelernte unmittelbar in der Praxis anzuwenden.

Neben den Grundlagen des Arduino Uno decken wir auch die Verwendung des leistungsstarken ESP32 ab, um Ihnen einen umfassenden Einblick in die Welt der Mikrocontroller zu zeigen. Sie werden Schritt für Schritt durch Anwendungen zur Temperaturregelung mit Fuzzy Logic geführt und lernen, wie Sie Motoren, sowohl in ihrer Drehzahl als auch in ihrer Position, regeln können.

Den Abschluss bilden ein Kapitel über Sliding-Mode-Regler sowie eines zu Grundlagen von Neuronalen Netzen, ebenfalls mit mehreren praktischen Beispielen.

Wir hoffen, dass dieses Buch nicht nur beim Lesen Freude bereitet, sondern Ihnen auch dabei hilft, Ihre eigenen Projekte erfolgreich umzusetzen. Wir wünschen Ihnen viel Spaß und Erfolg bei der Lektüre und bei Ihren zukünftigen Projekten!

Abschließend möchten wir allen danken, die uns unterstützt und Details immer wieder überprüft haben, die das Manuskript gelesen und Kommentare abgegeben haben, deren Anmerkungen wir zitieren durften und die uns bei der Redaktion, dem Korrekturlesen und der Gestaltung geholfen haben.

Josef Bernhardt

Kapitel 1 • Arduino Hardware für die Regelungstechnik

Arduino ist eine quelloffene Plattform für Elektronik-Projekte, die Hard- und Software kombiniert. Sie ist benutzerfreundlich und eignet sich sowohl für Anfänger als auch für erfahrene Entwickler. Es gibt viele verschiedene Arduino-Boards, Sensoren und Komponenten zur Auswahl.

Der Name Arduino stammt von einer Bar in der Stadt Ivrea in Italien, in der 2005 die Studenten Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino und David Mellis die Idee zur Entwicklung einer programmierbaren Platine für Prototypen hatten. Schnell wurde Arduino als Open-Source-Plattform veröffentlicht. Dies führte 2009 zur Gründung des Arduino-Unternehmens, das die Produktion und den Vertrieb der Boards unterstützt und kontinuierlich neue Produkte entwickelt.

Mit den Arduino Boards kann man Projekte in verschiedenen Bereichen realisieren, wie z.B. Robotik oder Industriesteuerungen. Für die Programmierung der Hardware Plattform gibt es die Arduino IDE, die das Schreiben, Testen und Hochladen von Programmen ermöglicht.

Die IDE basiert auf C/C++ und erleichtert Anfängern den Einstieg durch zahlreiche Bibliotheken und Beispielcode. Arduino bietet unendliche Möglichkeiten für Kreativität und Innovation.

In diesem Buch über Regelungstechnik kommen hauptsächlich der Arduino UNO und Arduino Nano sowie der ESP32 von Espressif zum Einsatz. Sehr interessant für komplexere Projekte ist auch der Raspberry RP2040.

1.1 Arduino UNO

Der Arduino UNO ist eines der beliebtesten Arduino-Boards und basiert auf dem ATmega328P-Mikrocontroller. Er bietet zahlreiche digitale und analoge Pins sowie Funktionen wie Analog/Digital Wandler, PWM-Ausgänge und serielle Kommunikation. Der UNO ist einfach zu bedienen und hat eine große Entwickler-Gemeinschaft, weshalb er ideal für die Regelungstechnik geeignet ist. Die Programmierung erfolgt über die Arduino-IDE mit C Code. Mit dem UNO können wir Sensoren, Heizungen, Motoren, LEDs und andere Komponenten steuern und hochwertige Regler aufbauen. Durch Erweiterungsplatinen lässt sich seine Funktionalität erweitern, etwa mit Displays, Netzwerk-Verbindungen oder drahtloser Kommunikation. Der UNO wird häufig in Bildungs- und Hobbyprojekten sowie in professionellen Anwendungen für Prototyping und schnelle Entwicklungen eingesetzt.

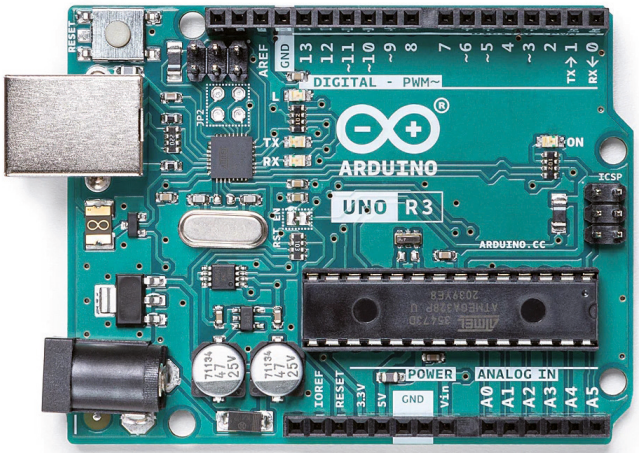


Abbildung 1.1: Arduino UNO Board

1.2 Arduino NANO

Sehr gut geeignet für Elektronikprojekte ist der Arduino Nano wegen seiner kompakten Bauform, der ebenfalls auf dem ATmega328P-Mikrocontroller basiert. Er bietet ähnliche Funktionen wie der Arduino UNO, jedoch in einem kleineren Formfaktor, was ihn ideal für platzsparende Projekte macht. Der Nano verfügt über 14 digitale I/O-Pins, 8 analoge Eingänge und unterstützt serielle Kommunikation sowie PWM-Ausgänge. Dank seiner geringen Größe und Flexibilität wird der Arduino Nano häufig in tragbaren und platzkritischen Anwendungen eingesetzt.

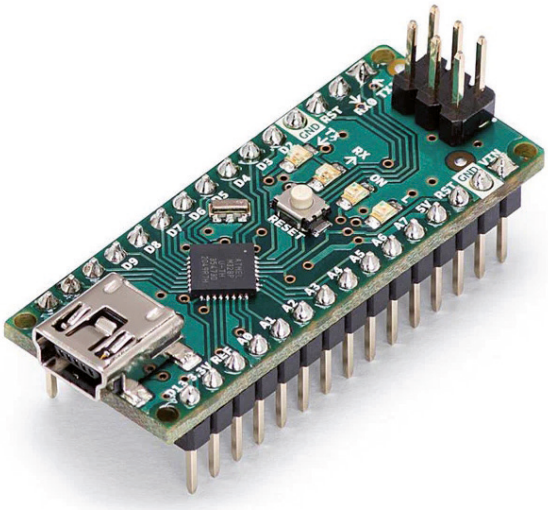


Abbildung 1.2: Arduino Nano Board

Merkmale und Spezifikationen der CPU

- Mikrocontroller: Atmel / Microchip ATmega328P
- Betriebsspannung: 5 Volt
- Eingangsspannung (empfohlen): 7-12 Volt
- Digitale I/O-Pins: 14 (davon 6 PWM-fähig)
- Analoge Eingangspins: 6 mit 10 Bit Auflösung
- Flash-Speicher: 32 KB (davon 0.5 KB für den Bootloader reserviert)
- SRAM: 2 KB
- EEPROM: 1 KB
- Taktfrequenz: 16 MHz

Arduino Nano Pin	Board	Beschreibung
D2	Taster S1	Digital Eingang aktiv High
D3	Taster S2	Digital Eingang aktiv High
D4	Taster S3	Digital Eingang aktiv High
D5	Taster S4	Digital Eingang aktiv High
D6	Taster S5	Digital Eingang aktiv High
D7	LED 1	Digital Ausgang
D8	LED 2	Digital Ausgang
D9	LED 6	PWM Ausgang
D10	LED 7	PWM Ausgang
D11	LED 8	PWM Ausgang
D12	LED 3	Digital Ausgang
D 13	LED 4	Digital Ausgang
ADC0	Poti R13	Analog Eingang
ADC1	Poti R14	Analog Eingang
ADC2	Poti R15	Analog Eingang
ADC3	Poti R16	Analog Eingang
ADC4	Poti R17	Analog Eingang

Tabelle 1.1: Pin Belegung Arduino Nano Testboard

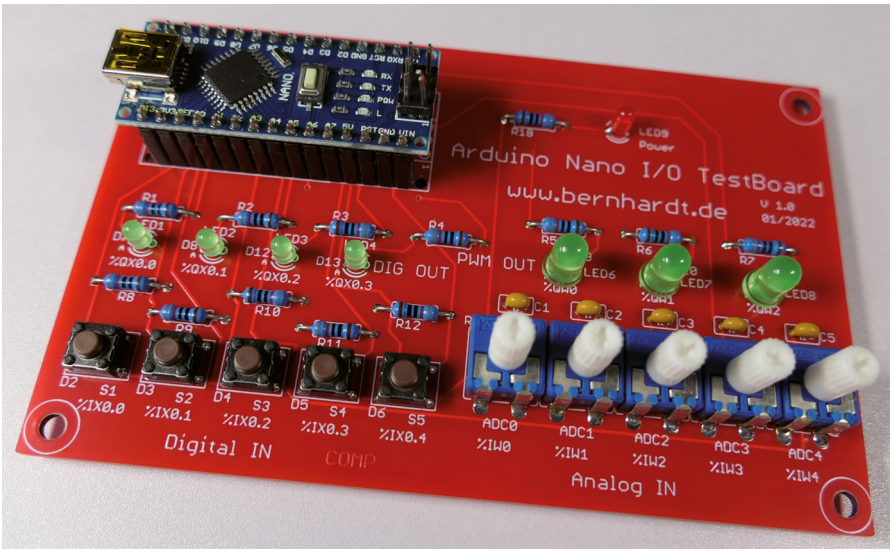


Abbildung 1.3: Arduino Nano Testboard

1.3 Espressif ESP32

Der ESP32 von Espressif ist eine leistungsstarke Entwicklungsplatine, die auf einem ESP32 Mikrocontroller basiert. Er bietet integriertes WLAN sowie Bluetooth, was ihn ideal für IoT-Anwendungen macht. Die Programmierung erfolgt hier ebenfalls über die Arduino-IDE, die von der offiziellen Arduino-Website [LINK1] heruntergeladen werden kann. Nach der Installation der ESP32-Boardunterstützung können wir den ESP32 mit C++-Code programmieren und auf seine Hardware zugreifen. Der ESP32 verfügt über zahlreiche GPIO-Pins für den Anschluss von Sensoren und Aktuatoren und kann sowohl als Server als auch als Client in WLAN-Netzwerken agieren. Zudem ermöglicht er die Kommunikation über Bluetooth mit anderen Geräten. Dank seiner Vielseitigkeit eignet sich der ESP32 für Projekte in den Bereichen Smart Home, Robotik und Regelungstechnik. Er ist auch für Anfänger geeignet und profitiert von einer großen Entwickler-Gemeinschaft sowie zahlreichen Online-Ressourcen, die beim Einstieg helfen. Mit dem ESP32 können wir Sensordaten sammeln und über WLAN oder auf einem TFT-Display anzeigen sowie Aktuatoren steuern.

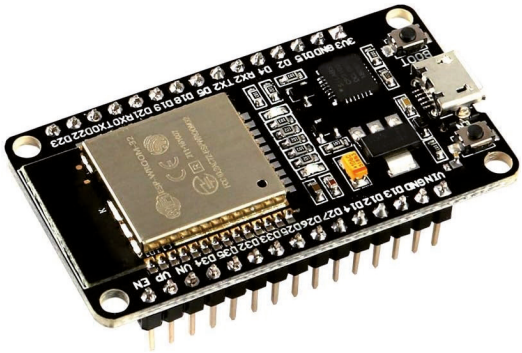


Abbildung 1.4: ESP32 Board

Hier die technischen Daten des ESP32 Mikrocontrollers

- Prozessor: Dual-Core Tensilica LX6 mit bis zu 240 MHz
- Speicher: RAM: 520 KB SRAM
  ROM: 448 KB
- Flash-Speicher: Je nach Modell, typischerweise 4 MB

Funktionen

- WLAN: 802.11 b/g/n
- Bluetooth: v4.2 BR/EDR und BLE
- GPIO-Pins: 34 Pins, vielseitig nutzbar für verschiedene Peripheriegeräte

Kommunikationsschnittstellen

- UART: 3 Schnittstellen
- SPI: 4 Schnittstellen
- I2C: 2 Schnittstellen
- I2S: 2 Schnittstellen
- PWM: bis zu 16 Kanäle
- DAC: 2 Kanäle
- ADC: bis zu 18 Kanäle, 12-bit Auflösung

Weitere Funktionen

- Touch-Sensoren: bis zu 10 kapazitive Touch-Eingänge
- RTC (Real-Time Clock)
- Hardware-Beschleunigung für Verschlüsselung (AES, SHA-2, RSA, ECC)

Energieverwaltung

- Verschiedene Energiesparmodi (Light Sleep, Deep Sleep)
- Betriebsspannung: 2.2 V – 3.6 V
- Betriebstemperatur: -40°C bis +85°C

Peripheriegeräte

- Hall-Sensor
- Temperatursensor
- Anschlussmöglichkeiten: USB, eine externe Stromversorgung ist über Batterie oder Akku möglich

1.4 Raspberry RP2040 Controller

Der RP2040 ist ein leistungsstarker Mikrocontroller, der von der Raspberry Pi Foundation entwickelt wurde. Er ist kostengünstig, basiert auf einer Dual-Core-Architektur und wurde speziell für Embedded-Anwendungen entwickelt. Er zeichnet sich durch seine hohe Leistungsfähigkeit, Flexibilität und Energieeffizienz aus.

Die CPU verfügt über eine Reihe von Funktionen, darunter GPIO-Pins, UART-, SPI- und I²C-Schnittstellen sowie eine Vielzahl von Peripheriemodulen. Er unterstützt auch die Programmierung über USB und verfügt über einen integrierten USB-Bootloader, der die Programmierung und den Betrieb ohne externe Programmiergeräte ermöglicht.

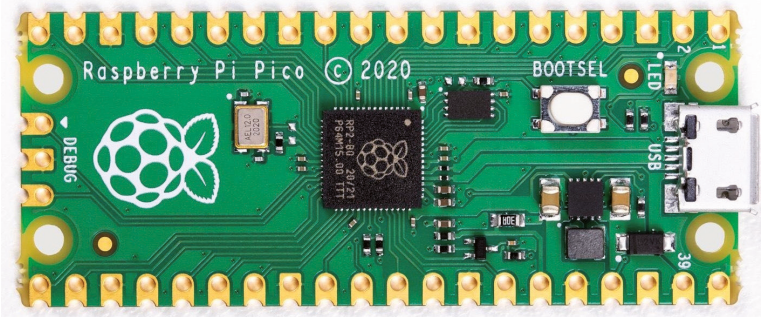


Abbildung 1.5: Raspberry RP2040 Board

Merkmale und Spezifikationen

- Mikrocontroller: Dual-Core ARM Cortex-M0+ (bis zu 133 MHz)
- Betriebsspannung: 3,3 Volt
- Digitale I/O-Pins: 30 (davon 26 PWM-fähig)
- Analoge Eingangspins: 4
- Flash-Speicher: 2 MB (QSPI)
- SRAM: 264 KB
- EEPROM: Nicht vorhanden
- Taktgeschwindigkeit: Bis zu 133 MHz

Der RP2040 wird von einer engagierten Entwicklergemeinschaft unterstützt, die eine Vielzahl von Ressourcen bereitstellt, darunter Dokumentationen, Tutorials, Foren und Beispielprojekte. Diese Ressourcen erleichtern die Entwicklung von Projekten und fördern den Austausch von Wissen und Erfahrungen in der Community.

1.5 ESP32 Fuzzy PID Controller Board

Für die Programmbeispiele in diesem Buch verwenden wir ein speziell für die Regelungstechnik entwickeltes Testboard. Die Platine "ESP32 Fuzzy PID Controller" ist eine vielseitige Plattform für präzise Regelungs- und Steuerungsanwendungen. Mit einer breiten Palette von Schnittstellen ermöglicht es eine nahtlose Integration in zahlreiche Anwendungen.

Weitere praktische Anwendungen wie der Einsatz als SPS oder Modbus TCP E/A Modul sowie Protokolle wie MQTT oder Webanwendungen mit Websockets sind jederzeit möglich.

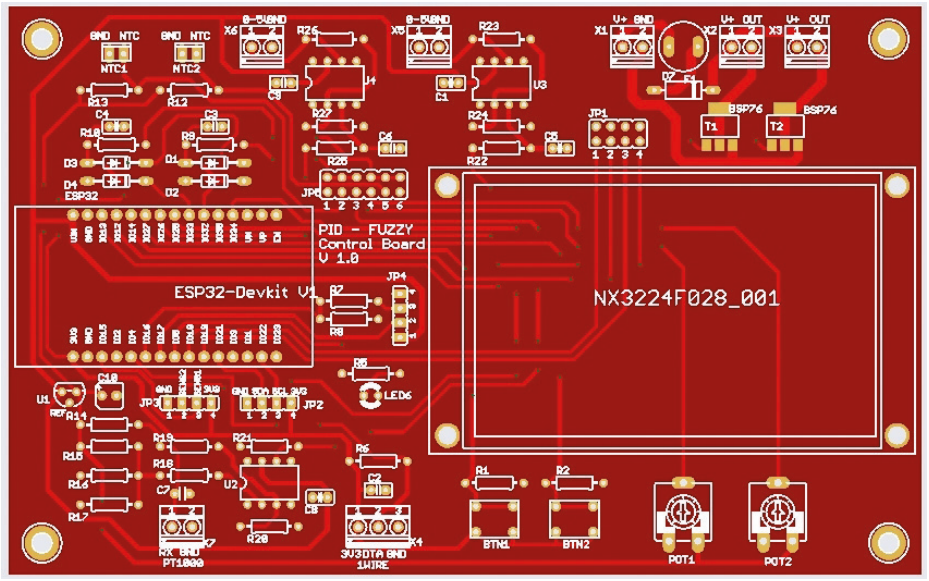


Abbildung 1.6: ESP32 Fuzzy PID Testboard

Eingangsschnittstellen

- 2 x NTC 10K: Diese Schnittstellen ermöglichen die genaue Messung von Temperaturänderungen durch NTC-Temperatursensoren.
- 1 x PT1000: Die PT1000-Schnittstelle ermöglicht die präzise Erfassung von Temperaturwerten mittels PT1000-Temperatursensoren.
- 1 x OneWire für DS18B20 oder DHT11 / DHT22: Diese flexible Schnittstelle unterstützt die Verwendung von DS18B20-Temperatursensoren oder DHT11 / DHT22 Feuchtigkeitssensoren für eine zuverlässige Umgebungserfassung.
- 2 Buttons: Zwei Tasten bieten eine einfache Benutzerschnittstelle für die Steuerung und Konfiguration des Systems.
- 2 Potis: Zwei Potentiometer ermöglichen die manuelle Einstellung von Parametern oder Schwellwerten für eine fein abgestimmte Steuerung.
- 2 Sense Leitungen für Motor Drehzahl und Positionserfassung: Diese Schnittstellen ermöglichen die präzise Erfassung der Motor-Drehzahl und -Position für eine genaue Regelung.

Ausgangsschnittstellen

- 2 x 24 V 1 A für Motor oder Relais wahlweise auch für PWM: Diese Ausgänge bieten die Flexibilität zur Steuerung von Motoren oder Relais, wahlweise über PWM-Signale für eine präzise Leistungsregelung.
- 2 x Analogausgang 0 bis 5 Volt: Zwei analoge Ausgänge ermöglichen die Erzeugung von Spannungssignalen im Bereich von 0 bis 5 Volt zur Ansteuerung externer Geräte.
- 1 x I2C-Anschluss 4-polig für Erweiterungen: Der I2C-Anschluss ermöglicht die einfache Integration von Erweiterungsmodulen für zusätzliche Funktionen oder Schnittstellen.

- 1 x UART2 TTL für ein Display Nextion NX3224F028 (2,8 Zoll): Das Nextion-Display bietet eine klare und benutzerfreundliche Oberfläche zur Anzeige von Daten und dank Touch auch zur Interaktion mit dem System.

Mit seinem leistungsstarken ESP32-Chip und einer Vielzahl von Schnittstellen bietet das **ESP32 Fuzzy PID Controller Board** eine robuste und flexible Lösung für eine Vielzahl von Anwendungen, von der Temperaturregelung bis hin zur Positionskontrolle.

Ein vollständiges Layout und die Schaltpläne befinden sich im Anhang.

Pinbelegung des ESP32 vom Testboard:

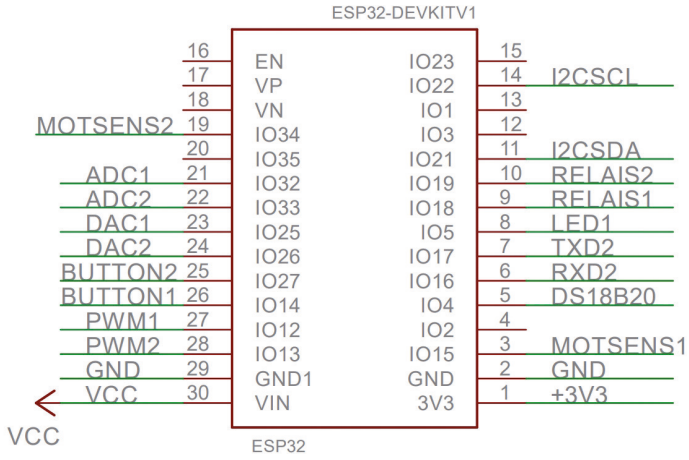


Abbildung 1.7: ESP32 Pinout

Pin Bezeichnung	ESP32 Modul Pin Nr.	GPIO Pin Nummer	Beschreibung
VCC	30		VCC Pin 5V vom USB
3V3	1		3.3 Volt Pin vom Spannungsregler
GND	2 / 29		GND Pin
ADC1	21	IO32	Analog Digitalwandler Kanal 1
ADC2	22	IO33	Analog Digitalwandler Kanal 2
BUTTON1	26	IO14	Taster 1 Eingang
BUTTON2	25	IO27	Taster 2 Eingang
MOTSENS1	3	IO15	Motor Sensor 1 Eingang
MOTSENS2	19	IO34	Motor Sensor 2 Eingang
I2CSDA	11	IO21	I2C Daten
I2CSCL	14	IO22	I2C Takt
RXD2	6	IO16	UART RX Eingang

TXD2	7	IO17	UART TX Ausgang
DS18B20	5	IO4	OneWire Schnittstelle
DAC1	23	IO25	Analog Ausgang 1 0 bis V3.3
DAC2	24	IO26	Analog Ausgang 2 0 bis V3.3
PWM1	27	IO12	Pulsweitenausgang 1
PWM2	28	IO13	Pulsweitenausgang 2
RELAIS1	9	IO18	Ansteuerung für Relais 1
RELAIS2	10	IO19	Ansteuerung für Relais 2
LED1	8	IO5	LED Board

Tabelle 1.2: Pinbelegung für das GPIO ESP32 Modul

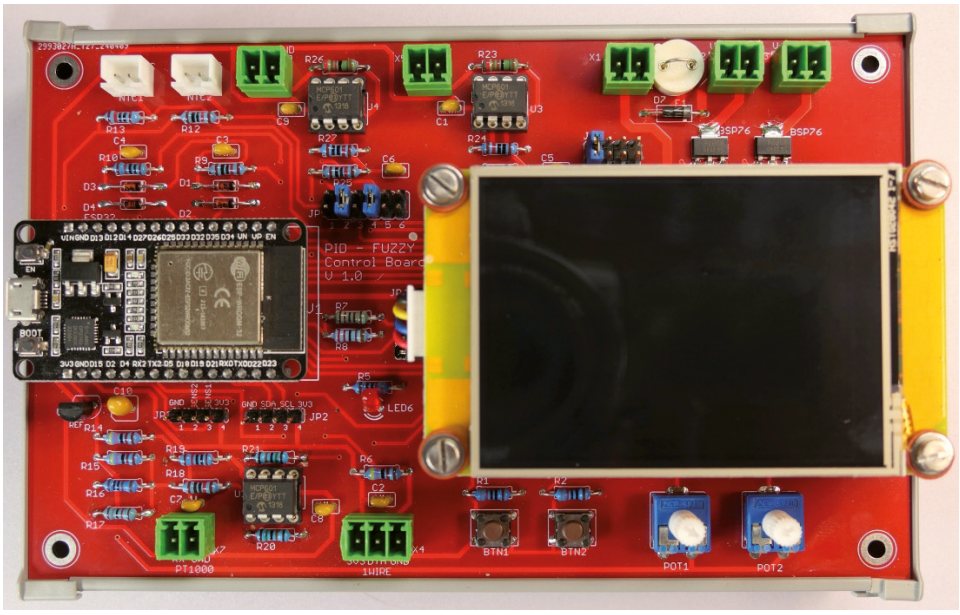


Abbildung 1.8: ESP32 Fuzzy PID Board

1.6 ESP32 Modbus RTU Controller Board

Ebenfalls geeignet, um Messwerte wie Strom, Spannung und Leistung erfassen zu können, ist das ESP32BoardModbusRTUControl Board.

Die Schaltpläne und Layouts dazu befinden sich im Anhang.

Das ESP32 Board wurde entwickelt, um Stromzähler, wie den **Shelly 3EM** über WLAN und die **Eastron SDM** Serie (baugleich mit **controlin**) sowie **KBR multimess 96,** über RS485 (Modbus RTU) auszulesen. Voraussetzung für den Betrieb ist ein funktionierendes

WLAN. Die Daten von den Zählern werden über das eingebaute Display angezeigt und im Sekundentakt per http im JSON Format an einen Server gesendet und dort gespeichert. XML wäre auch möglich. Die Controller Software wurde mit der Arduino IDE erstellt. Mögliche Protokolle sind http Post und Get, MQTT, REST, Modbus TCP Server und weitere, die mit TCP/IP möglich sind. Die Platine ermöglicht auch eine Ansteuerung von Wechselrichtern, um mit den Leistungsdaten eine Nulleinspeisung zu realisieren.

Technische Daten

CPU:	ESP32 von Tensilica
Stromversorgung:	12-24 Volt Gleichspannung
Schnittstellen:	RS485 WLAN 2,4 GHZ und Bluetooth UART RX TX für 5 V Nextion Display 2,4 Zoll NX3224T024 2 x 24V DC Ausgang für Relais oder Schütz. 1 x 10 V Analogeingang (erweiterbar auf 0-20 oder 0-50V) 1 x 10 V Analogausgang (oder 0-5V)1 x OneWire für Temperatursensor DS18B20

Infos zur Software des ESP32 Moduls

Das Modul verbindet sich nach dem Einschalten automatisch mit dem WLAN, dessen SSID und Passwort im Programm eingetragen sind. Sollte kein WLAN verfügbar sein, ist ein Access Point zu installieren.

```
// Wlan Parameter
char* ssid = "FRITZ!Box 7590 RI";
char* password = "42261153805385775967";
```

Danach hat man die Möglichkeit entweder über WLAN den Shelly 3 em auszulesen oder über Modbus RTU den Eastron SDM, oder den Controlin Zähler.

Hier ein Beispiel für die Abfrage des Shelly 3EM im Sekundentakt:

```
// Shelly 3 em
String serverName1 = "http://192.168.178.40/emeter/0";
// Antwort vom Shelly {"power":15.32,"pf":0.25,"current":0.27,"voltage":226.96,"is_valid":true,"total":2.0,"total_returned":0.0}
String serverName2 = "http://192.168.178.40/emeter/1";
// Antwort vom Shelly {"power":49.16,"pf":0.78,"current":0.27,"voltage":225.98,"is_valid":true,"total":5.9,"total_returned":0.0}
String serverName3 = "http://192.168.178.40/emeter/2";
// Antwort vom Shelly {"power":11.56,"pf":0.40,"current":0.13,"voltage":227.10,"is_valid":true,"total":2.0,"total_returned":0.0}
```

Die Daten werden vom JSON Format in Float-Werte konvertiert und die Gesamtleistung berechnet.

Danach wird von einem Timeserver der aktuelle Timestamp abgefragt und die Daten zum Firebase oder Postgre SQL Server gesendet.

```
// Firebase Server
String firebaseserver = "https://getup-now-37905-default-rtdb.europe-west1.firebasedatabase.app/customers/jbernhardt.json";

// http://93.90.178.80:3000/jbernhardt_sensor1
String postgreserver = "http://93.90.178.80:3000/jbernhardt_sensor1";

// {"sensor":"Shelly3em","timestamp":1676635242,"powerdata":[128.5298767,395.1396484,702.9299927,1226.599609]}
```

Die Ankopplung eines RS485 Zähler erfolgt über den UART1 und folgender Bibliothek:

https://github.com/reaper7/SDM_Energy_Meter

```
// RS485 Modbus RTU
#define RXD1 19
#define TXD1 18
#define REDE1 26

// Init RS485 Modbus RTO
SDM sdm(Serial1, SDM_UART_BAUD, REDE1, SERIAL_8N1, RXD1, TXD1); //config SDM
```

Es werden folgende Modbus Register ausgelesen:

```
PowerArr[0] = sdm.readVal(SDM_PHASE_1_POWER);
PowerArr[1] = sdm.readVal(SDM_PHASE_2_POWER);
PowerArr[2] = sdm.readVal(SDM_PHASE_3_POWER);
PowerArr[3] = sdm.readVal(SDM_TOTAL_SYSTEM_POWER);

PowerArr[4] = sdm.readVal(SDM_IMPORT_ACTIVE_ENERGY);
PowerArr[5] = sdm.readVal(SDM_EXPORT_ACTIVE_ENERGY);
```

Für Modbus RTU Zähler anderer Hersteller sind die Register anzupassen und Tests durchzuführen.

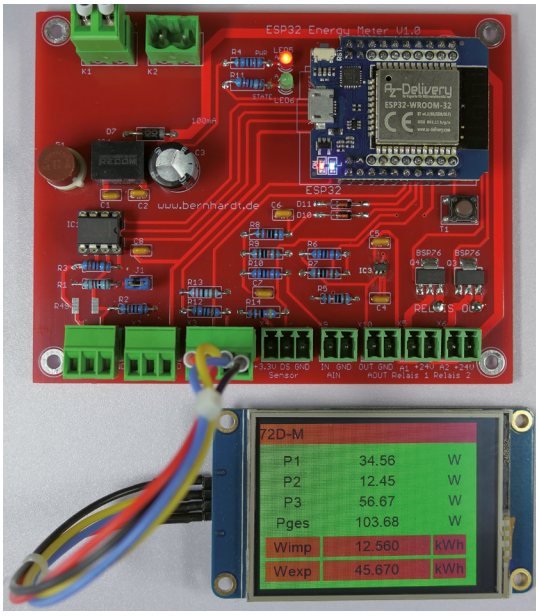


Abbildung 1.9: ESP32 Modbus RTU Board

Ebenfalls im Anhang sind die Schaltpläne und Layouts für ein Arduino Nano Board sowie für ein Arduino Mega Board mit jeweils vier Digital und Analog Leitungen.

1.7 Programmieren der Arduino Boards

Für die Programmierung der Boards steht eine Entwicklungsumgebung zur Verfügung die kostenlos von der Arduino Webseite heruntergeladen werden kann.

Die Software steht für Windows, Linux und macOS zum Download bereit.

Die Adresse der Wbseite: https://www.arduino.cc/en/software

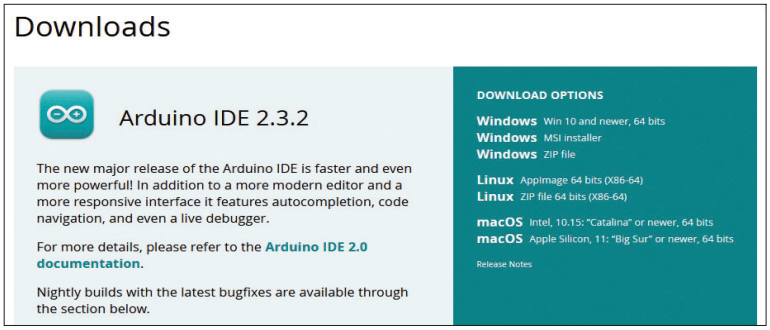


Abbildung 1.10: Download Arduino IDE

Nach der Installation der IDE können über die Boardverwalter Einstellung fehlende Boards wie das ESp32 nachinstalliert werden. Eine Liste der installierbaren Arduino Boards gibt es hier:

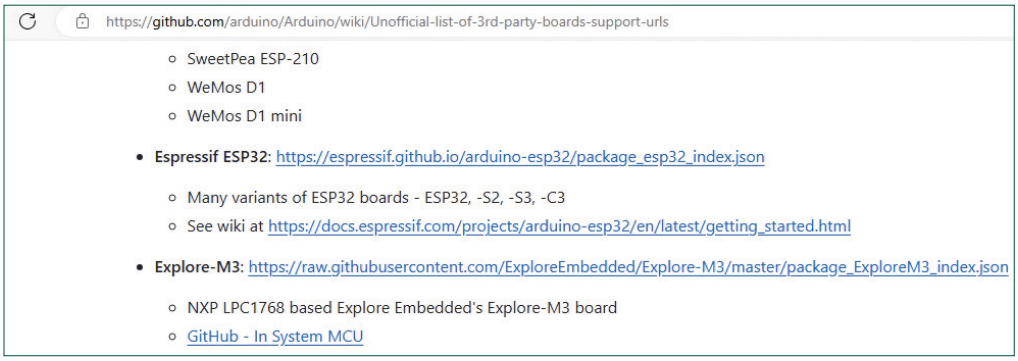


Abbildung 1.11: ESP32 Github Webseite

Unofficial list of 3rd party boards support urls · arduino/Arduino Wiki · GitHub

1. Öffnen Sie die Arduino IDE 2.3.2 auf Ihrem Computer.

2. Gehen Sie zu "Werkzeuge" > "Board" > "Boards-Verwalter".

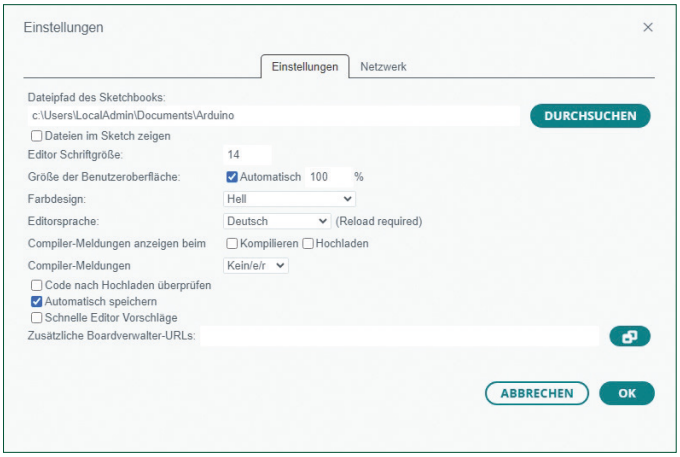


Abbildung 1.12: Einstellungen in der Arduino IDE

3. In der Suchleiste oben rechts im Boards-Verwalter "ESP32" eingeben und die Eingabetaste drücken.

4. Es sollte eine Liste der verfügbaren ESP32-Boards erscheinen. Wählen Sie das entsprechende ESP32-Board aus, das Sie verwenden möchten (z. B. "ESP32 Dev Module").

5. Klicken Sie auf den Installieren-Button neben dem ESP32-Board, um die Boardunterstützung zu installieren.

6. Warten Sie, bis die Installation abgeschlossen ist. Dies einige Minuten dauern.

7. Sobald die Installation abgeschlossen ist, kann man das ESP32-Board in der Liste der verfügbaren Boards unter "Werkzeuge" > "Board" auswählen.

Nachdem das ESP32-Board erfolgreich installiert ist, kann man es wie jedes andere Board in der Arduino IDE verwenden und Programme für den ESP32 schreiben und hochladen.

1.8 Das Nextion Grafik Display

Für die Anzeige der Messdaten in den Beispielen benutzen wir ein 2,8 Zoll Grafik Display NX3224F028_001 von der Firma Nextion. Der große Vorteil gegenüber anderen Displays, wie z.B. welche mit Grafikchip IL9341, ist die Ansteuerung mit dem UART über die RX/TX Leitungen und vor allem die einfachere Programmierung.

Für die Programmierung des Displays steht von der Firma Nextion eine kostenlose Entwicklungsumgebung für den PC zur Verfügung. Mit dieser kann man die Benutzeroberfläche für das Display am PC entwerfen. Es stehen viele Grafikelemente wie Buttons, Textboxen, Labels usw. zur Verfügung.

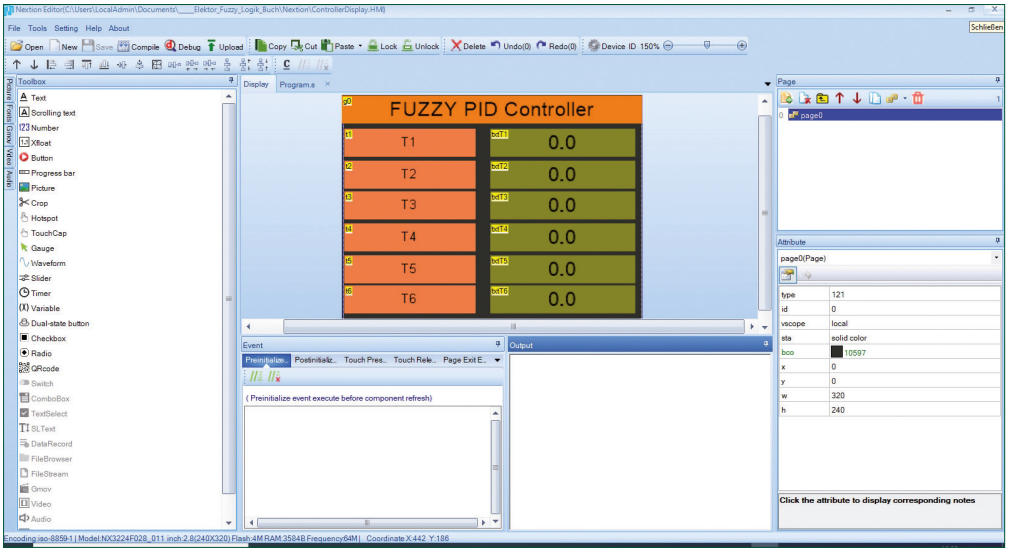


Abbildung 1.13: Nextion Editor IDE

Eine Debug-Funktion und eine Möglichkeit, die TFT-Datei auf das Display zu übertragen, sind ebenfalls in der IDE enthalten. Dazu kann man ein einfaches USB-UART Modul verwenden (Upload).

Eine zweite Möglichkeit ist, die erzeugte TFT Datei auf eine Micro-SD Karte zu speichern. Diesen Menüpunkt findet man unter "File" "TFT file output". Die SD-Karte wird dann in den SD Slot des Displays gesteckt und Spannung angelegt. Jetzt wird das Programm zum Mikrocontroller vom Display übertragen. Danach wird die Karte wieder entfernt. Dadurch erspart man sich das An- und Abstecken des Displays, wenn es auf der ESP32 Platine montiert ist.

Die Dateien ControllerDisplay.HMI und ControllerDisplay.tft sind im Verzeichnis "Nextion" gespeichert.

Auf der Webseite von Nextion finden sich viele weitere Infos und Tutorials.

1.9 Testsoftware für das ESP32 Board

In diesem Programm geht es um die Initialisierung der Hardware. Der Timer wird mit Hilfe der Ticker.h-Bibliothek auf 10 ms initialisiert und danach die beiden UART für die USB-Verbindung zum PC und UART2 für das Nextion Display.

Listing 1: Arduino_UART2_NextionDisplay.ino

```
//---------------------------------------------------------------------
//
//            Arduino Testsoftware
//            ====================
//
//      Testprogramm for Arduino ESP32 PID Fuzzy Control Board
//      10ms Timer Interrupt Function
//      Init UART2 for Nextion Display
//      Display seconds on Display NX3224F028_001
//
//  Date:     28/04/2024
//  Author:   Josef Bernhardt
//  Hardware: Arduino (ESPRESSIF) ESP32
//
//  File:     Arduino_UART2_NextionDisplay.ino
//---------------------------------------------------------------------
#include <Arduino.h>
#include <Ticker.h>

int LED_BUILTIN = 2;

// ESP32 UART 2 Pins
#define RXD2 16
#define TXD2 17

// I2C Pins
#define I2CSCL 22
#define I2cSDA 21

// ESP32 ADC Pins
#define ADC1 32
#define ADC2 33
// Motor Sensor Pins
#define MOTSENS1 15
#define MOTSENS2 34
// Buttons
#define BTN1 14
#define BTN2 27

// Relay and Mosfet (BSP76) Pins
#define RELAIS1 18
#define RELAIS2 19
// LED1
#define LED1 5
```

```
// PWM1 and PWM2 Pins
#define PWM1 12
#define PWM2 13
// DAC Outputs 0..5V
#define DAC1 25
#define DAC2 26
// One Wire
#define DS18B20 4
```

In diesem Code wird ein 'Ticker'-Objekt namens 'timer' verwendet, um eine Timer-Funktion alle 10 Millisekunden aufzurufen. Ein Zähler ('counter10ms') wird bei jedem Aufruf um 1 erhöht und wenn dieser Zähler den Wert 100 erreicht, was einer Sekunde entspricht, wird er zurückgesetzt. Die Variable 'OneSecond' wird auf 'true' gesetzt, um anzuzeigen, dass eine Sekunde vergangen ist. Der Sekunden-Zähler ('CounterSeconds') wird in 'loop()' verwendet um die Ausgabe jede Sekunde aufzurufen. Dabei wird OneSecond für den nächsten Durchlauf auf false gesetzt. So erhält man einen genauen Zeittakt für die Ausgabe des Sekundenzählers. Das Programm kann als "Grundgerüst" für die weiteren Programme verwendet werden.

```
// Define 10ms Period
const unsigned long timerInterval = 10;

// Ticker-Objekt for the Timer-Funktion
Ticker timer;

// Counter
int counter10ms=0;

// When 1 second is elapsed
bool OneSecond = false;

// Counter for seconds
int CounterSeconds = 0;

// Timer-Funktion
// Called every 10 ms
void timerCallback()
{
  // Inc 10ms Counter
  counter10ms++;
  // If one Second
  if(counter10ms==100)
  {
     counter10ms = 0;
     // One Second
     OneSecond = true;
```

```
  }
}
```

Diese Funktion sendet Daten an ein Nextion-Display, um den Text einer Textbox zu aktualisieren. Sie verwendet die serielle Schnittstelle 'Serial2', um einen Befehl zu senden, der die Textbox ('textbox') auswählt und den neuen Text ('sfloat') einfügt. Der Text wird in Anführungszeichen gesetzt und der Befehl wird mit drei Abschlussbytes ('0xFF') beendet, was das Ende des Befehls kennzeichnet. Dies ermöglicht es dem Nextion-Display, den Text korrekt anzuzeigen.

```
// Sends Data to Nextion Display in a textbox
void Nextion_Send_Text_to_Textbox(char *textbox,String sfloat)
{
        Serial2.write(textbox);
        Serial2.write(".txt=\"");
        Serial2.print(sfloat);
        Serial2.write("\"");
        Serial2.write(0xFF);
        Serial2.write(0xFF);
        Serial2.write(0xFF);
}

// Initializing the hardware
void setup()
{
  // Init USB UART
  Serial.begin(115200);
  // Nextion Display
  // TFTSerial.begin(9600);
  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
  // Init Hardware
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(RELAIS1, OUTPUT);
  pinMode(RELAIS2, OUTPUT);
  pinMode(BTN1, INPUT);
  pinMode(BTN2, INPUT);

  // Clear Textboxes
  Nextion_Send_Text_to_Textbox("t1","Counter");
  Nextion_Send_Text_to_Textbox("t2"," ");
  Nextion_Send_Text_to_Textbox("t3"," ");
  Nextion_Send_Text_to_Textbox("t4"," ");
  Nextion_Send_Text_to_Textbox("t5"," ");
  Nextion_Send_Text_to_Textbox("t6"," ");
```

```
  Nextion_Send_Text_to_Textbox("txtT2"," ");
  Nextion_Send_Text_to_Textbox("txtT3"," ");
  Nextion_Send_Text_to_Textbox("txtT4"," ");
  Nextion_Send_Text_to_Textbox("txtT5"," ");
  Nextion_Send_Text_to_Textbox("txtT6"," ");

  // Init Timer with 10ms
  timer.attach_ms(timerInterval, timerCallback);
}
```

In der 'loop()'-Funktion wird überprüft, ob eine Sekunde vergangen ist, basierend auf der 'OneSecond'-Variable. Der neue Wert von 'CounterSeconds' wird über die serielle Schnittstelle an den Seriellen Monitor gesendet und an eine Textbox auf dem Nextion-Display geschickt. Zudem wird der Zustand zweier LEDs umgeschaltet: die interne LED und eine weitere LED ('LED1'). Der Code sorgt so für regelmäßige Aktualisierungen und visuelles Feedback durch die LEDs.

```
void loop()
{

  // Check if one second has passed
  if (OneSecond) {
    // Reset the boolean variable OneSecond
    OneSecond = false;
    CounterSeconds++;

    // Sends variable to USB UART and Seriell Monitor
    Serial.print("Timer: ");
    Serial.println(CounterSeconds);

    // Sends variable to display
    Nextion_Send_Text_to_Textbox("txtT1",String(CounterSeconds));

    // Switching LED
    digitalWrite(LED_BUILTIN,!digitalRead(LED_BUILTIN));
    digitalWrite(LED1,!digitalRead(LED1));

  }
  // Additional code for the main loop
}
```

1.10 Programmbeispiel mit dem ESP32

Im nächsten Programm erfolgt eine Abfrage der Taster. Der Taster an BTN1 wird mit der Funktion 'read_btn1()' abgefragt und entprellt. Die Timer-Funktion wird jede 10 ms aufgerufen und der Taster an BTN2 eingelesen. Wenn dreimal hintereinander, also nach 30 ms, der BTN2 auf Low Pegel liegt, wird die Variable 'key_press' auf True gesetzt. Mit

der Funktion 'get_key()' wird der Tastendruck abgefragt. Mit den Informationen betätigt werden die Werte der Zustandsvariablen 'state1' oder 'state2' geändert, um den aktuellen Zustand zu speichern und dann die Ausgänge RELAIS1 und RELAIS2 zu schalten. Einmal drücken schaltet ein, nochmaliges Drücken wieder aus. Diese Funktionen werden später noch für die Einstellung des Sollwertes von Reglern gebraucht.

Listing 2: Arduino_TimerTick_USB_Hardware.ino

```
//-------------------------------------------------------------------
//
//            Arduino Timer Test
//            ==================
//
//      Timer Testprogramm for Arduino ESP32 PID Fuzzy Control Board
//      10ms Timer Interrupt Function
//      Read 2 Buttons and switch Relais Pin with BTN1 and LED1 with BTN2
//
//  Date:     08/05/2024
//  Author:   Josef Bernhardt
//  Hardware: Arduino (ESPRESSIF) ESP32
//
//  File:     Arduino_TimerTick_USB_Hardware.ino
//-------------------------------------------------------------------

#include <Arduino.h>
#include <Ticker.h>

int LED_BUILTIN = 2;

// ESP32 UART 2 Pins
#define RXD2 16
#define TXD2 17

// I2C Pins
#define I2CSCL 22
#define I2cSDA 21

// ESP32 ADC Pins
#define ADC1 32
#define ADC2 33
// Motor Sensor Pins
#define MOTSENS1 15
#define MOTSENS2 34
// Buttons
#define BTN1 14
#define BTN2 27
```

```
// Relay and Mosfet (BSP76) Pins
#define RELAIS1 18
#define RELAIS2 19
// LED1
#define LED1 5
// PWM1 and PWM2 Pins
#define PWM1 12
#define PWM2 13
// DAC Outputs 0..5V
#define DAC1 25
#define DAC2 26
// One Wire
#define DS18B20 4

// Define 10ms Period
const unsigned long timerInterval = 10;

// Ticker-Objekt for the Timer-Funktion
Ticker timer;

// Counter 10 for Timer
int counter10ms=0;

// When 1 second is elapsed
bool OneSecond = false;

// State for Buttons
int state1 = 0;
int state2 = 0;

// Variable for BTN2 in Timer
int key_state = 0;
int key_counter = 0;
int key_press = 0;
```

Neben der Zeitmessung wird in der 'TimerCallback'-Funktion der Zustand des Tasters BTN2 abgefragt. Um eine stabilere Tastererkennung zu gewährleisten, muss der Taster insgesamt mindestens 30 Millisekunden lang gedrückt sein. Bei Erreichen dieses Zählers wird der Tasterzustand aktualisiert und '`key_press`' auf 1 gesetzt, falls der Taster 30 ms gedrückt wird. Diese Methode stellt sicher, dass der Taster tatsächlich für eine bestimmte Zeit gedrückt wird, bevor er als gedrückt erkannt wird.

```
// Timer-Funktion
// Called every 10 ms
void timerCallback()
{
    // Inc 10ms Counter
    counter10ms++;
    // If one Second
    if(counter10ms==100)
    {
       counter10ms = 0;
       // One Second
       OneSecond = true;
    }

    // Read Button 2 in Timer function
    int input = digitalRead(BTN2);
    if( input != key_state )
    {
      key_counter++;
      if( key_counter == 3 )
      {
        key_counter = 0;
        key_state = input;
        if( input ) key_press = 1;
      }
    }
    else key_counter = 0;
}

// Read Button 2
int get_key()
{
      int result;
      result = key_press;
      key_press = 0;
      return result;
}
```

Die Funktion 'read_btn1()' prüft den Zustand des Tasters BTN1 und erkennt, ob er gedrückt wurde. Wenn der Taster von einem Zustand 'LOW' auf 'HIGH' wechselt (rising edge), wird ein kurzer Entprellungszeitraum von 20 Millisekunden eingehalten und der Zustand des Tasters erneut gelesen. Wenn der Taster weiterhin 'HIGH' ist, wird 'key' auf 1 gesetzt, um anzuzeigen, dass der Taster 20 ms gedrückt wurde. Schließlich wird der vorherige Tasterzustand für die nächste Überprüfung gespeichert und der Wert von 'key' zurückgegeben.