TUHH
Technische Universität Hamburg

Elektrotechnisches Versuchspraktikum

Für Bachelor-Studierende der Studiengänge Elektrotechnik (ET, 3. Semester), Elektrotechnik my Track (ET my Track, 5. Semester), Allgemeine Ingenieurwissenschaften (AIW 5. Semester), Engineering Science (5. Semester) und Technomathematik (TM 5. Semester)

Versuch Nr.: 3

Mikrocontroller

Ort: Gebäude Q, Raum 1.020

Allgemeine Informationen unter:

https://www.tuhh.de/mtec/teaching/e-praktikum

Stand der Versuchsbeschreibung: 12.10.2023

Versuch 3 (Mikrocontroller)

Versuchsbeschreibung vom 14.11.2022

Das Wichtigste in Kürze

Worum geht es?

In diesem Versuch wird der Mikrocontroller als universeller Baustein zur Realisierung von digitalen Funktionen zum Messen, Steuern, Anzeigen, Verarbeiten und Kommunizieren von Daten in elektronischen Geräten vorgestellt.

Was wird gemacht?

Mit der Arduino-IDE werden eigene Programme erstellt und auf dem Mikrocontroller zur Ausführung gebracht.

Welche Apparaturen und Instrumente werden verwendet?

Es werden ein ESP32 Mikrocontroller, ein LCD-Display, ein Luftfeuchtigkeits- und Temperatursensor, eine 32bit IO-Port-Erweiterung, eine LED-Zeile und die Arduino-IDE benutzt.

Was lernt man dabei?

Dass dieselben Mikrocontroller neben der Datenverarbeitung diverse Funktionen übernehmen können, angefangen von einfachen, logischen Verknüpfungen bis hin zur zeitgenauen Steuerung und Signalgenerierung, und dass der Hardware- und Software-Aufwand dafür gering ist.

Warum ist das wichtig?

Digitale Verarbeitungsfunktionen sind in nahezu jedem elektronischen Gerät anzutreffen, und der Ingenieur muss mit den Techniken und Bausteinen zu ihrer Realisierung vertraut werden. Die Digitaltechnik hat nicht nur viele ehemals in Analogtechnik realisierte Funktionen übernommen, sondern mit neuen Funktionalitäten erheblichen Mehrwert geschaffen. Innerhalb der Digitaltechnik können viele Anwendungen mit einer Standard-Hardware (Mikrocontroller) und einer flexibel anpassbaren Software preisgünstig und schnell realisiert werden.

Was wird von den Studenten erwartet?

Vollständige Durcharbeitung der Versuchsunterlagen und handschriftliche Beantwortung (digital ist ausreichend) der Kontrollfragen in Teil E.

Inhaltsverzeichnis

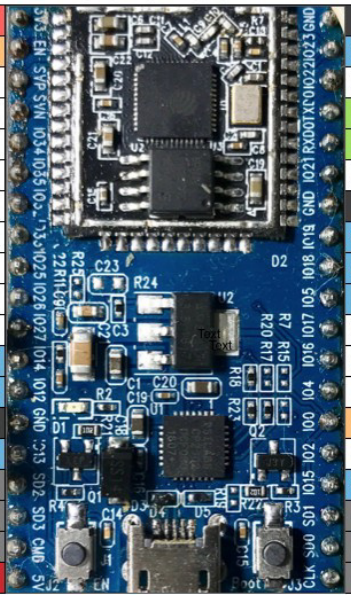
A Zielsetzung und Versuchsaufbau

In der heutigen Zeit, in der die Produktlebenszyklen immer kürzer werden, wird es für die Unternehmen immer wichtiger, eine möglichst kurze Time-to-Market zu realisieren. Dadurch lässt sich das für die Konkurrenzfähigkeit entscheidende Kosten-Nutzen-Verhältnis deutlich verbessern. Immer häufiger werden deswegen preiswerte Mikrocontroller benutzt, um zusammen mit Sensoren und Aktoren sowie Anzeige- und Bedien-Elementen aus Massenproduktion schnell und kostengünstig ein komplettes System zu realisieren. Dieses lässt sich zusätzlich auch noch per Firmware-Update umkonfigurieren.

Der Praktikumsversuch „Mikrocontroller“ soll mit dem Mikrocontroller als Bauelement digitaler Systeme bekannt machen.
Insbesondere wird gezeigt, wie der programmgesteuerte Mikrocontroller Messdaten erfassen und anzeigen kann. Es wird auch gezeigt, wie analoge Signale mit einem größeren Wertebereich von einem digitalen Prozessor erfasst und generiert werden können.
Bedenkt man, dass der Mikrocontroller seine Daten auch noch nahezu beliebigen Verarbeitungsschritten unterziehen kann, so wird die große Flexibilität des Mikrocontrollers als Systemkomponente deutlich.

In diesem Versuch wird das ESP32-DevKitC, ein ESP32 basiertes Entwicklungsboard, von Espressif Systems benutzt.

ESP32 Dev Board PINMAP

					3.3V		GND							
(pu)				RESET	EN		GPIO23	VSPI MOSI						SPI MOSI
SVP			ADC0		GPI 36		GPIO22							Wire SCL
SVN			ADC3		GPI 39		GPIO1	TX0						Serial TX
			ADC6		GPI 34		GPIO3	RX0						Serial RX
			ADC7		GPI 35		GPIO21							Wire SDA
		TOUCH9	ADC4		GPIO32		GND							
		TOUCH8	ADC5		GPIO33		GPIO19	VSPI MISO						SPI MISO
DAC1			ADC18		GPIO25		GPIO18	VSPI SCK						SPI SCK
DAC2			ADC19		GPIO26		GPIO5	VSPI SS					(pu)	SPI SS
		TOUCH7	ADC17		GPIO27		GPIO17							
	TMS	TOUCH6	ADC16	HSPI SCK	GPIO14		GPIO16							
(pd)	TDI	TOUCH5	ADC15	HSPI MISO	GPIO12		GPIO4		ADC10	TOUCH0		(pd)		
					GND		GPIO0	BOOT	ADC11	TOUCH1		(pu)		
	TCK	TOUCH4	ADC14	HSPI MOSI	GPIO13		GPIO2		ADC12	TOUCH2		(pd)		
				FLASH D2	GPIO9		GPIO15	HSPI SS	ADC13	TOUCH3	TDO	(pu)		
				FLASH D3	GPIO10		GPIO8	FLASH D1						
				FLASH CMD	GPIO11		GPIO7	FLASH D0						
					5V		GPIO6	FLASH SCK						

(Quelle: https://github.com/espressif/arduino-esp32/blob/master/docs/esp32_pinmap.png)

Der darauf enthaltene Mikrocontroller (ESP32) enthält zwei unabhängig voneinander programmierbare CPU-Kerne vom Typ "Xtensa LX6", mit einer
von 80 MHz bis 240 MHz einstellbaren Taktfrequenz.

Ebenfalls integriert sind:

-520 kB Ram
-4 MB Flash-Speicher
-34 x universelle Ein- und Ausgabeports (engl. GPIO – general purpose
-input/output)
-2 x 12-bit Analog-Digital-Wandler (ADC)
-2 x 8-bit Digital-Analog-Wandler (DAC)
-4 x Serial Peripheral Interface (SPI)
-2 × Inter-Integrated Circuit (I2C)
-1 x WiFi / WLAN / Bluetooth Modul

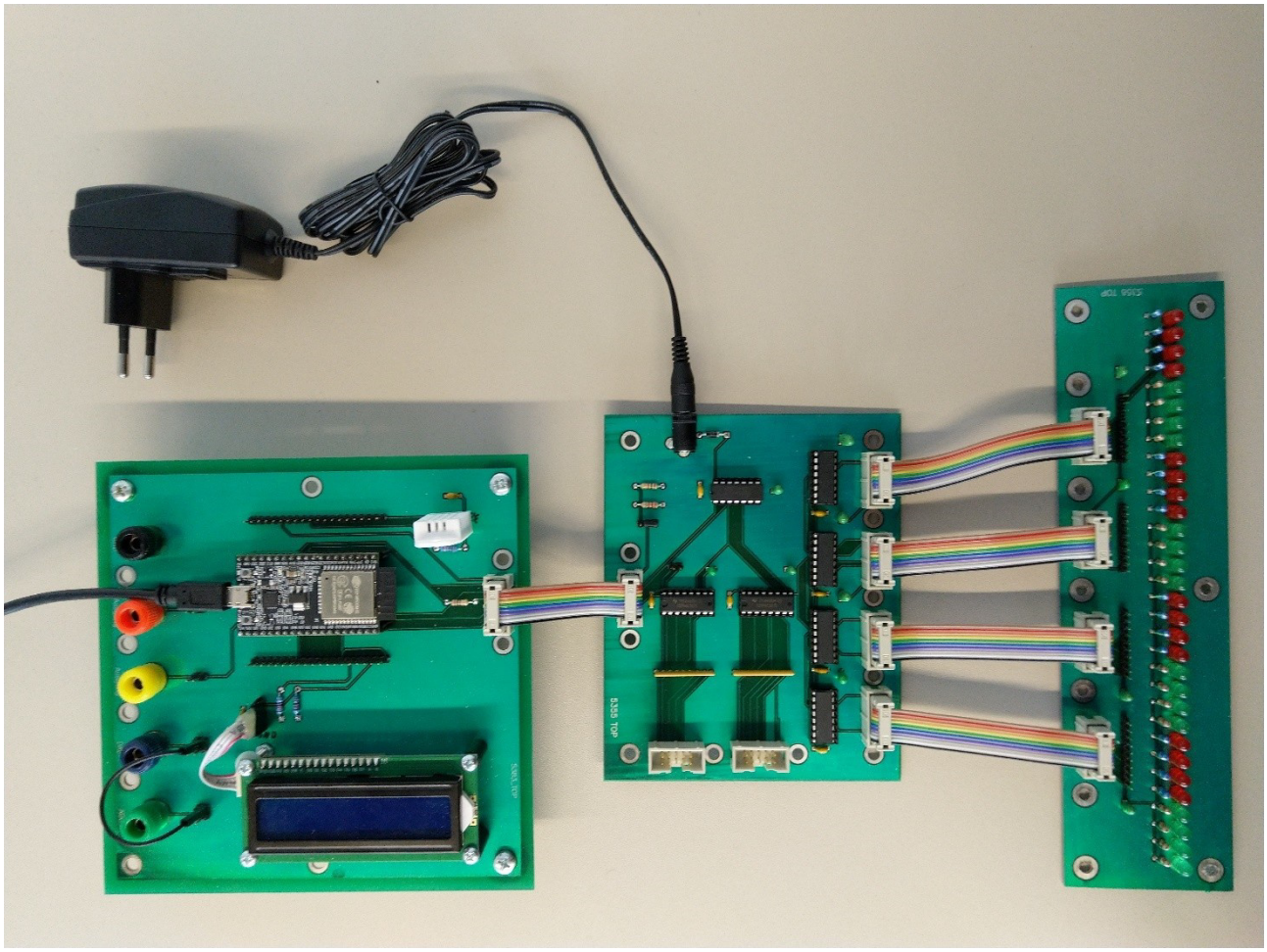


Bild vom Versuchsaufbau (v. l. n. r.: Mainboard, I/O-Port Erweiterung und LED-Zeile)

Der Versuchsaufbau besteht aus drei über Flachbandkabel miteinander verbundenen Platinen:

1 Mainboard:

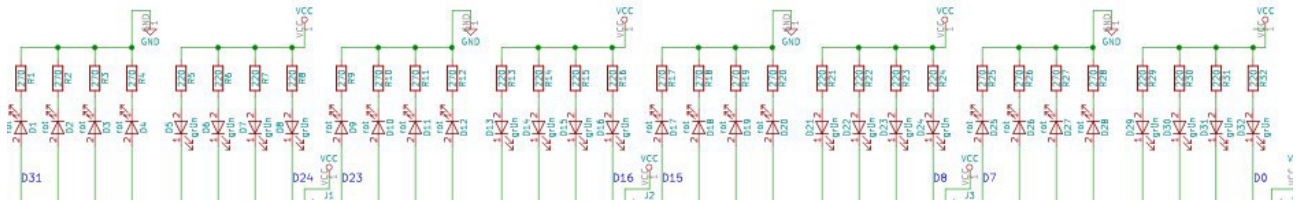
Auf dem Mainboard mit aufgestecktem ESP32-DevC Modul befinden sich zusätzlich auch noch ein LCD-Display und ein Temperatur- und Luftfeuchtigkeitssensor vom Typ DHT22. Das LCD-Display ist über den I2C-Bus und der DHT22 über einen Single-Bus angeschlossen.
Der HSPI-Bus des ESP32 ist auf einem 10-poligen Wannenstecker herausgeführt. Des Weiteren sind Bananenbuchsen für GND, 5V und 3,3V vorhanden. Zusätzlich sind die GPIOs 35 und 25 (ADC und DAC) des ESP32 auf zwei Bananenbuchsen herausgeführt.

2 I/O-Port Erweiterung:

Die über den SPI-Bus angesteuerte 32bit I/O-Port Erweiterung wird über eine externe 5V Spannungsquelle gespeist. Diese Platine wurde mit Schieberegistern der Typen „74HC595“ und „74HC597“ realisiert.
Zusätzlich zu den üblichen SPI-Bus Signalen ist das Signal „Output-Enable“, mit welchem sich die Ausgänge in den Tri-State-Zustand versetzen lassen, auf dem 10-poligen Wannenstecker vorhanden. Die Platine enthält sechs zusätzliche 10-polige Wannenstecker, welche 16 digitale Eingänge und 32 digitale Ausgänge verfügbar machen.

3 LED-Zeile:

Die LED-Zeile dient zur Anzeige der in den Schieberegistern gespeicherten Werte. Sie besteht aus 32 LEDs nebst passendem Vorwiderstand und wird über die vier integrierten Wannenstecker angesteuert.
Die LEDs sind in farblich gut zu unterscheidenden Vierergruppen (Nibbles) aufgeteilt. Die Hälfte dieser Vierergruppen wird invertiert angesteuert, um den Stromfluss durch die Schieberegister zu optimieren.



Schaltplan der LED-Zeile

B Zusätzliche Informationen

B.1 Zahlenformate

Wie allgemein bekannt sein sollte arbeiten moderne Computer binär, das heißt nur mit 1en und 0en. Um mit den zwei Zuständen auszukommen, wird in der ALU eines jeden Prozessors mit Binärzahlen gerechnet. Dieser Umstand kann ausgenutzt werden, um damit eine einfache Repräsentation von Zuständen umzusetzen. Für dieses Praktikum repräsentiert jedes Bit einer 32 Bit Ganzzahl den Zustand, den die LED's auf der LED-Zeile annehmen sollen. Wenn Zahlen in C(++) Code geschrieben werden, interpretiert der Compiler diese standardmäßig als Dezimalzahlen. Diese Darstellung ist aber sehr unglücklich, wenn man einzelne Bits oder Nibbles (4 Bit Blöcke) ansprechen möchte. Soll z.B. die dritte und siebte LED leuchten, läuft dies auf eine Rechnung hinaus, welche nicht mehr ganz so einfach im Kopf durchgeführt werden kann. Nämlich: $2^2+2^6=4+64=68$. Die Verringerung des Exponenten um 1 kommt daher, dass davon ausgegangen wird, dass die LED's mit 1 beginnend, Bits aber bei 0 beginnend gezählt werden. Für dieses Beispiel wirkt das noch einigermaßen simpel, aber man stelle sich die Rechnung für eine 32 Bit Zahl vor in der z.B. 20 Bits gesetzt werden.
Viel einfacher ist hier die Rechnung bzw. Zahlenschreibweise im Binär- oder Hexadezimalsystem. Hier beschreibt eine Stelle immer genau den Wert eines Bits bzw. Nibbles. Praktischerweise gibt es in vielen Programmiersprachen die Möglichkeit Zahlen direkt in diesen Systemen im Code zu schreiben, indem Präfixe vor eine Zahl geschrieben werden. Für das Binärsystem wird 0b, und für das Hexadezimalsystem wird 0x verwendet. Die Rechenaufgabe vom vorherigen Absatz vereinfacht sich somit zum einfachen Schreiben von 1en und 0en an den richtigen Stellen. Diese folgenden Ausdrücke sind alle äquivalent:

$$2^2 + 2^6 = 64 = \texttt{0b01000100} = \texttt{0x44}$$

B.2 Logische Verknüpfungen in C/C++

C(++) bietet einige Arten von logischen Verknüpfungen, welche sich fundamental in Logische und Bitweise Verknüpfungen unterteilen. Die grundlegenden Wahrheitstabellen können u.a. der Vorlesung Technische Informatik entnommen werden. Dabei ist die Nomenklatur wie folgt:

Verknüpfung	Operator Bitweise	Operator Logisch
AND	&	&&
OR	|	||
XOR	^	!= (bool)
INVERT	~	!

Bei Logischen Verknüpfungen werden Variablen oder Ausdrücke vor der Verknüpfung nach Wahr oder Falsch ausgewertet. Das Ergebnis der Verknüpfung `0b0011 && 0b1010` ist somit `true AND true`, also wahr, da beide Zahlen ungleich null sind. Der Ausdruck `0b0000 && 0b1010` wertet sich zu `false AND true` aus, also falsch.

Bei der Bitweisen Verknüpfung hingegen, welche jede Bitposition einzeln betrachtet, kann man Zahlen dazu verwenden, um andere zu manipulieren. Diese Programmiertechnik wird im Umfeld der Embedded Systems oder auch Treiberprogrammierung häufig verwendet, um Registerwerte aus Einstellungen zu bilden oder zu manipulieren. Ein großer Vorteil ist, dass mit diese Bitweisen

Operationen, je nach Hardwarearchitektur und verwendete Datentypen, in einzelne Assembler Befehle umgesetzt und somit sehr effizient ausgeführt werden können. Im einfachsten Fall hat man dabei eine Eingangszahl oder Variable (im Folgenden Ganzzahl `A`) und eine Maske (Ganzzahl `mask`), welche auf die Zahl angewendet wird. Das Ergebnis (Ganzzahl `B`) ist dann die bitweise Verknüpfung der Zahl mit der Maske. Anschaulich kann man die Verknüpfungen ähnlich einer schriftlichen Addition/Subtraktion ausführen. Zunächst werden A und mask als Binärzahl übereinandergeschrieben, und dann unter einem Strich das Ergebnis notiert wie z.B. hier:

```
  0b01111001
& 0b00001111
  ----------
  0b00001001
```

Genauso wie bei anderen Operatoren (z.B. +=, -=, …) kann in C(++) auch die Verkürzte Form verwendet werden, sofern Eingangsvariable und Ergebnis die gleiche Variable sind. Folgende Funktionen können über bitweise logische Verknüpfung realisiert werden:

- Bits einer Zahl „ausschneiden“ oder auch maskieren: Bitweise AND
  - `B = A & mask;` bzw. `A &= mask;`
  - Alle Bits, welche in der Maske 0 sind, werden auf 0 gesetzt, alle welche in der Maske 1 sind behalten den Wert von A
- Bits auf 1 setzen: Bitweise OR
  - `B = A | mask;` bzw. `A |= mask;`
  - B hat an allen Bitpositionen eine 1 an denen A oder die Maske eine 1 hat. Somit können Bits in A „angeschaltet“ werden
- Bits Toggeln/Umschalten: Bitweise XOR
  - `B = A ^ mask;` bzw. `A ^= mask;`
  - Bitwert von A wir an allen Stellen invertiert, an denen die Maske eine 1 aufweist
- Bits auf 0 setzen: Spezialform bitweise AND
  - `B = A & ~mask;` bzw. `A &= ~mask;`
  - Spezialform des bitweise AND bei dem an allen Bitpositionen, welche 1 in der Maske sind eine 0 geschrieben wird und alle anderen unverändert bleiben.

Ergänzend sollte noch erwähnt werden, dass Bitmanipulation unabhängig von Variablentypen mit allen Werten im Speicher funktioniert. So kann z.B. mit einer passenden Maske der Exponent einer float Variablen auf 0 gesetzt werden.

B.3 Bit Shifting

Zusammen mit den bitweisen logischen Verknüpfungen ist das bit Shifting ein wichtiger Teil der Bitmanipulation. Soll ein Bitmuster in Variable A um n Stellen nach links oder rechts geschoben werden, kann dies über folgenden Ausdruck geschehen:

`B = (A<<n);` bzw. `B = (A>>n);`

Im Grunde kann man einen Shift nach links als eine Multiplikation mit 2^n bzw. nach rechts mit 2^{-n} ansehen, welche allerdings bei der Ausführung meist deutlich effizienter als eine Multiplikation umgesetzt werden kann. Aufpassen muss man, dass es zwei verschiedene Arten von Shifts gibt: Den logischen und arithmetischen Shift. Der Logische Shift ist ein reines Schieben und füllt an den Rändern immer nullen auf, der arithmetische setzt die 2^n Multiplikation vorzeichenrichtig für Ganzzahlen um, füllt also beim Schieben nach rechts auf der linken Seite den Bitwert auf, welcher

vorher im MSB stand. Standard im gcc Compiler ist der arithmetische Shift. Soll der logische verwendet werden muss man als Datentypen eine vorzeichenlose (unsigned) Variable verwenden.

C Details zum Versuchsaufbau

Zur Programmentwicklung benutzen wir die im Internet frei verfügbare Arduino-IDE, eine komfortable Plattform für die Entwicklung von Embedded- und IoT- Anwendungen.
Mithilfe der Arduino-IDE und den ebenfalls gratis im Internet erhältlichen Bibliotheken für zahlreiche Sensoren und Peripheriegeräte lassen sich schnell und ohne großen Aufwand fertige Anwendungen realisieren.

Die Programme (Sketches) werden auf dem PC in einem C/C++ -Dialekt entwickelt und für den ESP32 kompiliert. Über eine USB-Verbindung wird ein erfolgreich übersetztes Programm in den Flash-Speicher des ESP32 übertragen und dort gestartet. Dort arbeitet es dann in einer Endlosschleife.

Um auf dem Mikrocontroller laufen zu können, muss ein Sketch immer eine bestimmte Grundstruktur aufweisen. Es muss eine Funktion „void setup()“ enthalten. Diese wird zu Beginn des Programmablaufs nur einmal ausgeführt. Hier werden typischerweise die Initialisierungen vorgenommen. Des Weiteren muss eine Funktion „void loop()“ enthalten sein, welche vom Mikrocontroller nach der Ausführung von „void setup()“ kontinuierlich wiederholt wird und die eigentliche Anwendung enthält. Oberhalb der Funktion „void setup()“ werden üblicherweise noch Bibliotheken eingebunden und globale Variablen deklariert.
Die Arduino-IDE gibt diese Struktur beim Erstellen eines neuen Programmes bereits vor:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

C.1 Installation der Arduino-IDE

Der folgende Teil dient nur dazu, um die Konfiguration im Versuchsraum nachbilden zu können. Auf den PC's im Versuchsraum ist die Installation der Arduino-IDE bereits durchgeführt. Die Installation auf Ihrem eigenen Rechner ist nicht notwendig.

Bitte installieren Sie zunächst die portable Version der Arduino-IDE.
(Es besteht auch die Möglichkeit, Ihnen eine funktionsfähige Version der Software zur Verfügung zu stellen. Diese Methode hat allerdings den Nachteil, dass dann das Desktopsymbol fehlt und die IDE in der Konsole gestartet werden muss.)

Starten Sie dazu das Terminal und geben Sie die folgenden Befehle ein:

```
mkdir Versuch_3

cd Versuch_3
```

Sie benötigen die aktuelle Version der Arduino-IDE. Laden Sie diese von der Arduino Homepage herunter:

https://www.arduino.cc/en/Main/Software

Wählen Sie die 64bit-Version für Linux aus. Die Dateien werden im Ordner „Downloads" gespeichert. Mit den folgenden Befehlen erzeugen Sie ein passendes Installationsverzeichnis und installieren die Software:

```
cp ../Downloads/arduino-1.8.7-linux64.tar.xz .

tar xvf arduino-1.8.7-linux64.tar.xz

rm arduino-1.8.7-linux64.tar.xz

cd arduino-1.8.7/

mkdir portable

cd portable

mkdir packages

mkdir staging

mkdir sketchbook

cd sketchbook/
```

```
mkdir libraries

cd

cd Versuch_3

cd arduino-1.8.7/

./install.sh
```

Der letzte Befehl erzeugt auch ein Symbol auf dem Desktop. Damit ist die Installation der Arduino-IDE abgeschlossen und Sie können mit der Installation des ESP32 Arduino Core und der für den Versuch benötigten Bibliotheken beginnen.

C.2 Installation des ESP32 Core und der Bibliotheken

Starten Sie die Arduino-IDE über das Desktopsymbol und führen Sie folgende Schritte aus:

Über "Datei"->"Voreinstellungen" auf das Symbol rechts neben dem Eingabefeld für "ZusätzlicheBoardverwalter-URLs" klicken.

"https://dl.espressif.com/dl/package_esp32_index.json" eingeben und mit OK bestätigen.

Unter "Werkzeuge"->"Board"->"Boardverwalter" esp32 suchen und installieren und anschließend das Fenster schließen.

Unter „Werkzeuge“ → „Board:“ ESP32 Dev Module auswählen.

Unter „Sketch“ → „Bibliothek einbinden“ → „Bibliotheken verwalten“ nach „ESP32“ suchen und „DHT sensor library for ESPx by BEEgEE_tokyo“ installieren.

Unter Typ: "Alle" und Thema: „Display“ nach "I2C" suchen und "LiquidCrystal I2C by Frank de Barbander" installieren.

"Schließen".

C.3 Beispielprogramm für die Benutzung des LCD

Jetzt sollten Sie in der Lage sein, Ihren ersten Sketch zu programmieren. Machen Sie dazu bitte Folgendes:

Den ersten Sketch anlegen unter: „Datei“ → „Speichern unter...“

Sketch mit „LCD_Beispiel“ benennen und speichern.

Geben Sie anschließend den folgenden Programmcode ein:

```
//Die Bibliothek "LiquidCrystal_I2C" einbinden
#include <LiquidCrystal_I2C.h>

// LCD-Adresse 0x27, auf einem 16 Zeichen und 2 Zeilen Display
LiquidCrystal_I2C lcd(0x27,16,2);

void setup() {
  float ver=1.0;
  int i=3;

  // das LCD initialisieren
  lcd.init();

  // die Hintergrundbeleuchtung einschalten
  lcd.backlight();

  // Schreibposition Zeichen 0 , Zeile 0
  lcd.setCursor(0,0);

  // Text ausgeben
  lcd.print("  e-praktikum   ");

  // Schreibposition Zeichen 0 , Zeile 1
  lcd.setCursor(0,1);

  lcd.print(" Versuch ");

  // int ausgeben
  lcd.print(i);

  lcd.print(" V");

  // float mit einer Nachkommastelle ausgeben
  lcd.print(ver,1);

 // Programm pausieren, für 5000 Millisekunden
  delay(5000);

  // Programm pausieren, für 5 Mikrosekunden
 delayMicroseconds(5);

  // Inhalt des LCD löschen
  lcd.clear();
}

void loop() {
}
```

Stellen Sie die USB-Verbindung zwischen dem PC und dem Mainboard her und wählen Sie unter „Werkzeuge“ → „Port“ /dev/ttyUSB0 aus.

Das Programm wird mit dem „Hochladen“ (Pfeil nach rechts in der zweiten Menüleiste) kompiliert, auf das Mainboard geladen und anschließend gestartet.

In seiner Setup-Funktion beinhaltet dieses Beispielprogramm bereits alle für die Durchführung des Versuchs notwendigen Befehle zur Ansteuerung des Displays und zum Verzögern des Programmes.

D Teilversuche

D.1 Hygrometer

Für die Ermittlung der Schimmelgefahr in Wohnräumen werden häufig Hygrometer eingesetzt. Sie messen die Lufttemperatur und die relative Luftfeuchtigkeit. Aus diesen Werten lässt sich dann der Taupunkt errechnen.
Als Taupunkt bezeichnet man die Temperatur unterhalb welcher Wasserdampf zu kondensieren beginnt. In Wohnräumen ist es deshalb wichtig, dass auch an der kältesten Stelle der Taupunkt nicht unterschritten wird.
Auf dem Mainboard ist bereits ein Temperatur- und Luftfeuchtigkeitssensor vom Typ DHT22 integriert. Die bereits installierte Bibliothek „DHTesp“ bietet komfortable Befehle, um aus diesem Sensor die Temperatur und Luftfeuchtigkeit auszulesen. Auch für die Berechnung des Taupunktes ist eine Funktion vorhanden.

Schreiben Sie ein Programm, welches die relative Luftfeuchtigkeit und die Temperatur aus dem Sensor DHT22 ausliest und zusammen mit dem Taupunkt auf dem LCD ausgibt. Beachten Sie dabei, dass der Sensor zwei Sekunden benötigt, um seine Werte zu aktualisieren.

D.1.1 Beispiel für die Benutzung des DHT22

```
//Die Bibliothek "DHTesp" einbinden
#include "DHTesp.h"

//Ein neues DHTesp-Objekt mit dem Namen dht erzeugen
DHTesp dht;

// Pinnummer des DHT22 Datenpins
// Der Sensor ist an GPIO27 des ESP32 angeschlossen
#define dhtPin 27

void setup(void){
  // Temperatursensor initialisieren
  dht.setup(dhtPin, DHTesp::DHT22);
}

void loop(void) {

// Objekt vom Typ struct TempAndHumidity erzeugen
TempAndHumidity lastValues;
float TP;

// Auslesen von Temperatur und Luftfeuchtigkeit
  lastValues = dht.getTempAndHumidity();

// Taupunkt berechnen
  TP = dht.computeDewPoint( lastValues.temperature,
lastValues.humidity);
}
```

D.2 Voltmeter

Schreiben Sie ein Programm namens „Voltmeter“, welches die Werte von 0 bis 255 auf dem Digital-Analog-Converter (DAC) ausgibt, nach jedem Wert 10 ms wartet und die daraus resultierende Spannung mit dem Analog-Digital-Converter (ADC) wieder einliest. Geben Sie das Ergebnis der Analog-Digital-Wandlung auf dem LCD aus (in Zeile 0).

Berechnung der Spannung aus dem ADC Wert:

$$Spannung = ADCwert \cdot \frac{Referenzspannung}{Maximalwert}$$

Die Auflösung des ADC beträgt 12bit und die Referenzspannung ist 3,28V. Berechnen Sie mit diesen Werten die Spannung in Volt und geben Sie diese auf dem LCD aus (in Zeile 1).

D.2.1 Beispiel für die Benutzung des ADC und des DAC

```
// ADC
// Den Eingang GPIO35 des ESP32 benutzen
#define  analogPin 35

// DAC
// Den Eingang GPIO25 des ESP32 benutzen
#define dacPin 25

void setup(){
}

void loop(void){
// Funktion zum Ausgeben eines Wertes auf den DAC
void dacWrite(dacPin,  int )

// Funktion zum Einlesen eines Wertes vom ADC
int analogRead(analogPin)
}
```

D.3 Ansteuerung der I/O-Port Erweiterung

Für die Bearbeitung dieser Aufgabe ist zunächst die I/O-Port Erweiterung an den SPI-Bus der Hauptplatine anzuschließen. Die Porterweiterung wird zusätzlich über das Steckernetzteil mit 5V versorgt und zur Anzeige der Ausgangszustände mit der LED-Zeile verbunden.

Das Serial Peripheral Interface (SPI) stellt einen „lockeren“ Standard für einen synchronen seriellen Datenbus (*Synchronous Serial Port*) dar, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können.
(Quelle: https://de.wikipedia.org/wiki/Serial_Peripheral_Interface)

Der SPI-Bus besteht aus den zwei Datenleitungen MOSI (Master Output->Slave Input) und MISO (Master Input<-Slave Output), der Taktleitung (SCLK) und Chip-Select-Leitung (CS). Die SPI-Hardware des ESP32 generiert diese Signale automatisch.
Damit der Zustand der Schieberegister an die Ausgänge durchgeschaltet wird, ist zusätzlich OEPIN auf HIGH zu setzen.

D.3.1 Beispiel für die Benutzung des SPI-Busses

```
#include <SPI.h> // SPI-Bibliothek einbinden

#define OEPIN 4 // OUTPUT ENABLE

#define PLPIN  23 //  Latch Clock

// SPISettings Objekt mit 8 MHz Takt erzeugen
SPISettings settingsA(8000000, MSBFIRST, SPI_MODE0);

// 32 Bit auf dem SPI-Bus ausgeben
void spi_out( long wert )
{
// SPI Bus mit den Werten aus settingsA initialisieren

  SPI.beginTransaction(settingsA);


  digitalWrite(PLPIN, HIGH); //PLPIN auf HIGH setzen

  SPI.transfer32( wert );  // 32 Bit übertragen

  /* mit einer fallenden Flanke am PLPIN (Latch Clock)
     werden die Werte aus dem Schieberegister in das
     Storage Register (Latch) übertragen. */
  digitalWrite(PLPIN, LOW); //PLPIN auf LOW setzen
```

```
  digitalWrite(PLPIN, HIGH); //PLPIN auf HIGH setzen

  SPI.endTransaction(); // SPI Bus freigeben

}

void setup(void)
{
  // OEPIN als Ausgang konfigurieren
  pinMode(OEPIN, OUTPUT);

  // OEPIN auf HIGH setzen
  digitalWrite(OEPIN, HIGH);


  // PLPIN als Ausgang konfigurieren
  pinMode(PLPIN, OUTPUT);

  // PLPIN auf HIGH setzen
  digitalWrite(PLPIN, HIGH);

  // HSPI-Bus initialisieren
  SPI.begin(14, 12, 13, 15); // sck, miso, mosi, ss

}
void loop(void)
{
   long muster;
  spi_out( muster );

}
```

D.3.2 Binärzähler

Schreiben Sie dazu zunächst eine Funktion, welche einen long Wert so umcodiert, dass seine Einsen eine Leuchtdiode auf der LED-Zeile zum Leuchten bringen.
Testen Sie diese Funktion, indem Sie einen 32bit Wert von Null beginnend bis 0xffffffff hochzählen lassen und dabei die jeweiligen Zählerstände auf der LED-Zeile darstellen.

Anmerkung! Die Verarbeitungsgeschwindigkeit der verwendeten Hardware ist sehr hoch. Der Programmablauf ist mit der Funktion delay() zu verzögern, damit das menschliche Auge die einzelnen Ausgaben erfassen kann.

D.3.3 Bargraph-Anzeige

Schreiben Sie ein Programm, welches die Leuchtdioden nacheinander einschaltet und dann alle gemeinsam wieder ausschaltet.

D.3.4 Kraftfahrzeugblinker

Wandeln Sie anschließend dieses Programm so um, dass es einen den gesetzlichen Vorgaben entsprechenden dynamischen Fahrtrichtungsanzeiger (als Bargraph-Anzeige) implementiert.

Die gesetzlichen Vorgaben sind:

> Eine Frequenz von 1,5 Hz ± 0,5 Hz (90 Lichterscheinungen pro Minute ± 30)
>
> Eine Taktung der Blinkgeber so, dass sie eine relative Hellzeit der Blinkleuchten von 30 % bis 80 % liefern.

D.4 Multiprocessing und PWM

Damit die LED-Zeile auch bei wechselnden Lichtverhältnissen, beispielsweise in Wohnräumen, eingesetzt werden kann, ist es sinnvoll, die Helligkeit der Leuchtdioden einstellbar zu machen.

In diesem Versuch bietet es sich an, dazu die Pulsweitenmodulation (PWM) zu verwenden. Sie wird häufig zur verlustarmen Steuerung von Verbrauchern mittels digitaler Signale eingesetzt. Dazu wird bei gleichbleibender Frequenz ein Rechtecksignal in seinem Tastgrad (Tastgrad = Einschaltdauer / Periodendauer) so eingestellt, dass ein gewünschter zeitlicher Mittelwert entsteht. Wenn dabei die Frequenz hoch genug ist, wird ein angeschlossener Verbraucher (Motor, Lautsprecher, …) durch seine eigene Trägheit die Mittelwertbildung, also eine Demodulation, vornehmen. Ein Vorteil dabei ist auch, dass die Leistungstransistoren immer voll durchgeschaltet oder voll sperrend betrieben werden, wodurch an ihnen nur wenig Verlustleistung abfällt.

Wenn das Output-Enable-Signal (OEPIN) der Schieberegisterplatine in seiner Pulsweite moduliert wird, lässt sich damit, unabhängig von der sonstigen Funktion der Platine, die Helligkeit der Leuchtdioden einstellen.

Anmerkung! Die Leuchtdioden in unserem Versuch sind so schnell, dass eine Mittelwertbildung dort nicht stattfindet. Allerdings betrachten wir sie mit unseren Augen, welche träge genug sind, um dann den Mittelwert bilden.

Um die beiden Prozessoren und die integrierte Peripherie zu verwalten, benutzt Espressif das Betriebssystem FreeRTOS. FreeRTOS unterstützt asymmetrisches Multiprocessing (AMP). Mit nur wenig zusätzlichem Aufwand lassen sich damit an eine bestimmte Recheneinheit gebundene Prozesse erzeugen. Es bietet sich an, die Pulsweitenmodulation auf dem bisher im Versuch

unbenutzten CORE1, also der zweiten Recheneinheit des ESP32, zu implementieren. Ein solcher Prozess (Task) wird üblicherweise am Ende der setup Funktion gestartet.

D.4.1 Beispiel für die Benutzung von mehreren Prozessen

```
void dimmTask( void * pvParameters ){
// PWM an OEPIN ausgeben
}
void setup(void)
{
  // Task erzeugen
  xTaskCreatePinnedToCore(
    dimmTask,   // Name der Funktion
    "dimmTask", // Name der Task
    10000,      // Stackgröße in Worten
    NULL,
    0,         // Priorität 0
    NULL,
    1);        // CORE 1
}
```

D.4.2 LED-Zeile dimmen

Erweitern Sie das Blinkerbeispiel aus Aufgabe 3 mit dem zusätzlichen Prozess „dimmTask“. Dieser soll eine PWM erzeugen, deren Tastgrad der am ADC gemessenen Spannung entspricht. Die PWM ist auf dem OEPIN auszugeben. Die Frequenz soll ungefähr 244Hz betragen.

Hinweise:

> Zum Einlesen, Ausgeben und Anzeigen eines Analogwertes kann die aus Aufgabe 2 bekannte Funktion verwendet werden.
>
> Der Programmablauf kann mit der Funktion delayMicroseconds() verzögert werden.
>
> Der dimmTask wird nur einmal aufgerufen. In der Funktion muss also dafür gesorgt werden, dass die Software PWM in einer Endlosschleife ausgeführt wird.

E Kontrollfragen

Welche Binärzahl muss an der LED-Zeile anliegen, damit alle Leuchtdioden leuchten?

Was versteht man unter dem Tastgrad (Dutycycle) einer PWM?

Wie werden die Ausgänge der I/O-Port Erweiterung aktiviert?

Füllen Sie die folgenden Wahrheitstabellen aus:

&	0	1
0		
1		

|		

^		

Führen Sie die folgenden bitweisen Verknüpfungen schriftlich durch:

```
int B = 16 | 32;
```

```
int C = 0b10110010 & 0b00110011;
```

```
int D = 0b001110101 ^ 0xf0;
```

Es sollen die im Kommentar genannten LED-Zustände in die Variable `leds` als C Code geschrieben werden. Tipp: verwenden Sie auch das Binär- oder Hexadezimalsystem.

```
int leds =                          ; //Alle LEDs aus
int leds =                          ; //1.& 5. LED an
int leds =                          ; //Alle LEDs an
int leds =                          ; //LED 1-4 & 9-15 an
```