1. Arbeiten mit VS-Code Portable und Platform IO

1.1. Was ist PlattformIO[1]

PlatformIO ist eine integrierte Entwicklungsumgebung (IDE) und ein Ecosystem für die Entwicklung von Software für eingebettete Systeme (Embedded Systems).

Es ist speziell für die Arbeit mit Mikrocontrollern und IoT-Geräten konzipiert und bietet eine Vielzahl von Funktionen, die den Entwicklungsprozess erleichtern. Hier sind einige wichtige Merkmale von PlatformIO

- Unterstützung für mehrere Plattformen und Frameworks: PlatformIO unterstützt eine große Anzahl von Mikrocontroller-Plattformen wie Arduino, ESP8266, ESP32, STM32, Atmel AVR, Nordic nRF und viele mehr. Es unterstützt auch verschiedene Entwicklungsframeworks wie Arduino, mbed, CMSIS, Zephyr und FreeRTOS.
- Cross-Plattform-Kompatibilität: PlatformIO ist auf verschiedenen Betriebssystemen wie Windows, macOS und Linux verfügbar, was es Entwicklern ermöglicht, plattformunabhängig zu arbeiten
- Integration in verschiedene IDEs: PlatformIO kann als eigenständige Anwendung verwendet werden oder in gängige Entwicklungsumgebungen wie Visual Studio Code, Atom und CLion integriert werden. Die Integration in Visual Studio Code ist besonders beliebt, da sie eine leistungsfähige, leichtgewichtige und benutzerfreundliche IDE bietet.
- Automatisches Bibliotheksmanagement: PlatformIO bietet ein robustes System zur Verwaltung von Bibliotheken, dass die automatische Installation und Aktualisierung von Abhängigkeiten unterstützt. Entwickler können einfach die benötigten Bibliotheken hinzufügen, und PlatformIO kümmert sich um
- Build-System und Debugging: PlatformIO verfügt über ein leistungsfähiges Build-System, das auf einem Einheitlichen Skript basiert und für verschiedene Plattformen angepasst werden kann. Außerdem unterstützt es erweiterte Debugging-Funktionen, die bei der Fehlersuche in der Firmware helfen
- Open-Source und Community-getrieben: PlatformIO ist eine Open-Source-Software mit einer aktiven Community, die regelmäßig zu ihrer Verbesserung beiträgt. Dadurch werden ständig neue Funktionen und Plattformen hinzugefügt.

[1] https://platformio.org/

1.2. VS-Code Installation

Besuchen Sie die offizielle Visual Studio Code Website:

1. Rufen Sie die Seite https://code.visualstudio.com/ in Ihrem Browser auf.
2. Wählen Sie die portable Version aus (funktioniert im Unterrichtraum meist proble,loser. Ist aber Geschmackssache.)
3. Um die portable Version herunterzuladen, wählen Sie die ZIP-Datei (nicht den Installer) aus.
4. Um Visual Studio Code im portablen Modus zu betreiben, erstellen Sie einen Ordner mit dem Namen data im Hauptverzeichnis von Visual Studio Code (dort, wo sich die ausführbare Datei Code.exe befindet).
5. Dieser Ordner data wird für alle Benutzerdaten und Erweiterungen verwendet, sodass keine Daten auf dem Computer gespeichert werden.

Vorteile des portablen Modus:

- Alle Daten, einschließlich Einstellungen und Erweiterungen, bleiben im data-Ordner gespeichert.
- Ideal für den Einsatz auf USB-Sticks oder anderen externen Laufwerken.

1.3. PlatformIO-Extension installieren

Um PlatformIO mit Visual Studio Code (VS Code) zu nutzen, benötigen Sie in erster Linie die PlatformIO IDE Erweiterung. Diese Erweiterung enthält alles, was Sie für die Entwicklung von Embedded-Software mit PlatformIO brauchen. Die Erweiterungen für C/C++ , CMake, CMake Tools werden dann auch mitinstalliert.

- erstellen Sie einen Ordner für Ihren Workspace

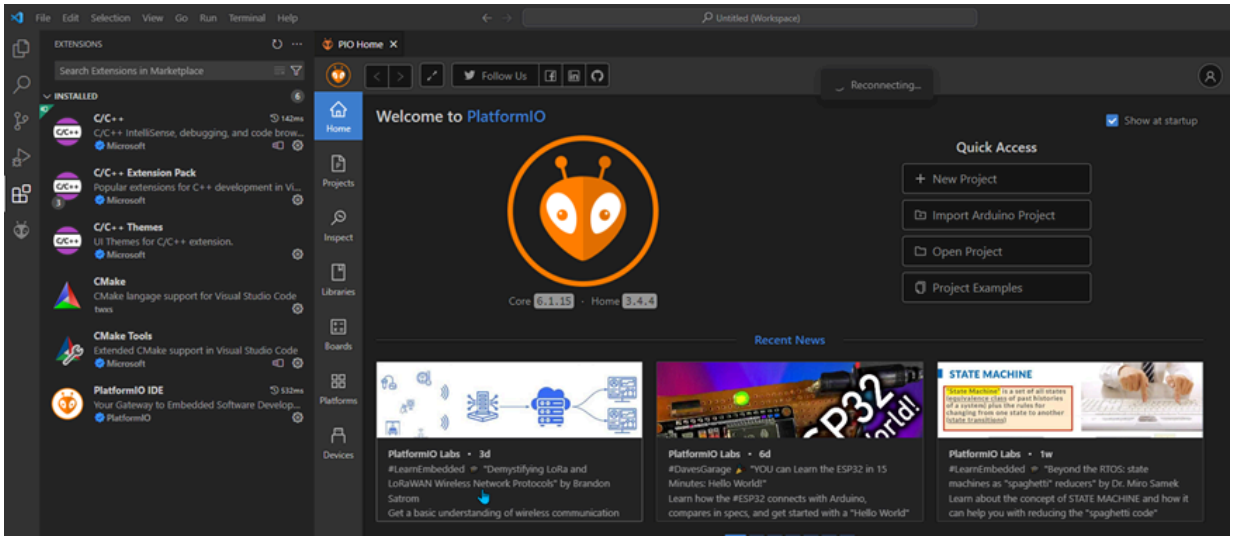

Abbildung 2: Benötigte Erweiterung für VSCode

1.4. Projekt im Workspace Odner erzeugen

Gehen sie auf sas Alien Symbol → dann im Quick Access Open → dann im Window PIO Home auf new Project

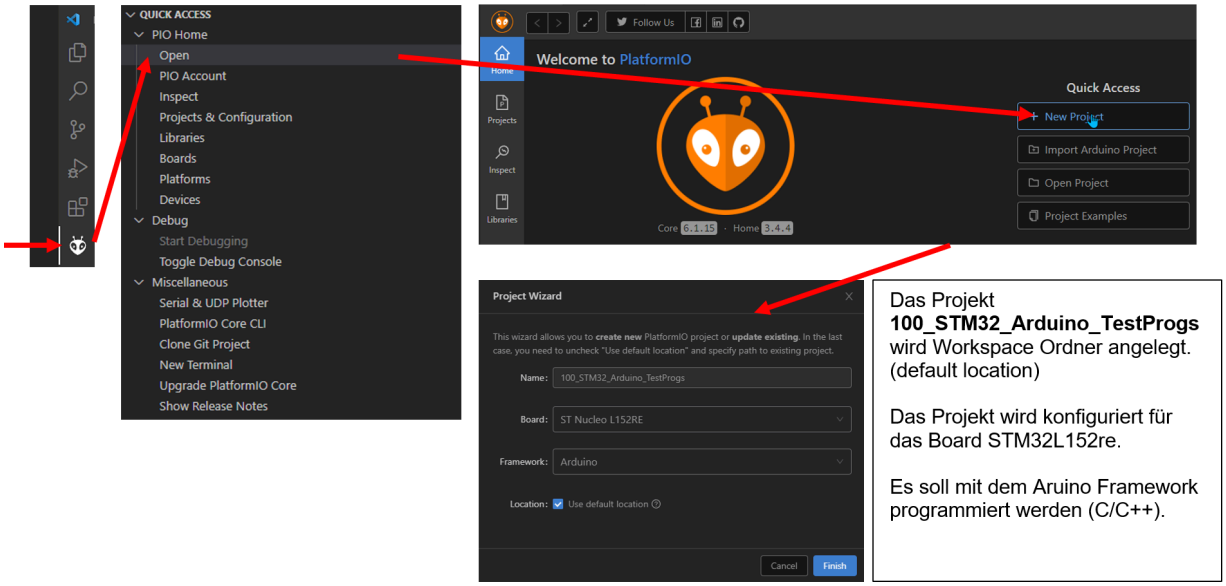

Abbildung 3: Projekt mit PlatformIO anlegen

1.5. Projektstruktur in PlatformIO

Projektname ist der Stammordner des Projekts. Der Name des Ordners kann beliebig sein, sollte aber in der Regel den Projektnamen widerspiegeln.

```
Projektname/
├── .pio/
├── .vscode/
├── include/
├── lib/
├── src/
├── test/
├── platformio.ini
└── README.md
```

1.5.1. .pio/

Beschreibung: Dieser Ordner wird von PlatformIO automatisch generiert und enthält alle Build-Artefakte, Zwischenspeicher, und temporären Dateien.

Inhalt: Verschiedene Unterordner und Dateien, die vom Build-System verwendet werden, einschließlich kompilierten Binärdateien, Object Files, und mehr.

Hinweis: Dieser Ordner sollte in der Regel nicht manuell bearbeitet werden und ist oft in .gitignore enthal-ten, um ihn von der Versionskontrolle auszuschließen.

1.5.2. .vscode/

Beschreibung: Enthält Konfigurationsdateien für die VS Code Entwicklungsumgebung. **Inhalt:** Dateien wie c_cpp_properties.json, launch.json, und settings.json, die verwendet werden, um Intel-liSense, Debugging und andere Editor-Einstellungen in VS-Code zu konfigurieren. **Hinweis:** Diese Dateien werden oft automatisch generiert, können aber bei Bedarf manuell angepasst werden.

1.5.3. include/

Beschreibung: Dieser Ordner ist für Header-Dateien (.h oder .hpp) vorgesehen, die global im Projekt ver-fügbar sein sollen.
Inhalt: Benutzerdefinierte Header-Dateien, die von mehreren Quellcode-Dateien im Projekt verwendet werden.
Hinweis: Verwende diesen Ordner, um Header-Dateien zu organisieren, die in mehreren Modulen oder Quellcode-Dateien eingebunden werden.

1.5.4. lib/

Beschreibung: Dieser Ordner ist für externe Bibliotheken bestimmt, die in deinem Projekt verwendet werden.
Inhalt: Jeder Unterordner innerhalb von lib/ entspricht einer separaten Bibliothek. Jede Bibliothek kann ihre eigene Struktur mit Header-Dateien und Quellcode-Dateien haben.
Hinweis: PlatformIO bietet auch eine Möglichkeit, Bibliotheken automatisch zu verwalten und herunterzu-laden, aber du kannst auch manuell Bibliotheken in diesem Ordner platzieren.

1.5.5. src/

Beschreibung: Enthält den Hauptquellcode deines Projekts.
Inhalt: Quellcode-Dateien wie main.cpp (oder main.c, abhängig von der verwendeten Programmiersprache) und alle weiteren .cpp- oder .c-Dateien, die den Kern deines Projekts bilden.
Hinweis: PlatformIO sucht standardmäßig in diesem Ordner nach den Hauptquellcode-Dateien, die kompiliert werden sollen.

1.5.6. test/

Beschreibung: Dieser Ordner ist für Unit-Tests vorgesehen. **Inhalt:** Testskripte und -dateien, die zum Testen von Modulen oder Funktionen innerhalb deines Projekts verwendet werden. **Hinweis:** PlatformIO unterstützt die Entwicklung von Unit-Tests und das Testen von Embedded-Software. Testskripte können je nach Testszenario strukturiert werden.

1.5.7. platformio.ini

Beschreibung: Die Hauptkonfigurationsdatei für dein PlatformIO-Projekt.

Inhalt: Diese Datei enthält Projektkonfigurationen wie die Zielplattform (z.B. STM32, ESP32), Board-Spezifikationen, Build-Flags, Bibliotheksabhängigkeiten, und andere Einstellungen.

Hinweis: Die platformio.ini ist das Herzstück deines PlatformIO-Projekts, da sie bestimmt, wie Plat-formIO den Code kompiliert und welche Einstellungen verwendet werden.

```
; -------------  Konfiguriert die allgemeinen Einstellungen -----------
[platformio]
; Definiert das Verzeichnis für die kompilierten Build-Dateien
build_dir = C:/BUILD_DIR

; --------------  Konfiguriert die Umgebung für das NUCLEO-L152RE -----
[env:nucleo_l152re]
platform = ststm32 ; Setzt die Plattform auf ST STM32
board = nucleo_l152re  ; Spezifiziert das Board: NUCLEO-L152RE
framework = arduino  ; Verwendet das Arduino-Framework für STM32

; -------------- Konfiguriert die Umgebung für das NUCLEO-F103RB ------
;[env:nucleo_f103rb] ; Definiert ein weiteres Environment für das NUCLEO-F103RB
;platform = ststm32 ; Setzt die Plattform auf ST STM32
;board = nucleo_f103rb ; Spezifiziert das Board: NUCLEO-F103RB
;framework = arduino  ; Verwendet das Arduino-Framework für STM32

;---------- Konfiguriert die allgemeinen Einstellungen für alle Umgebungen  -----
; Verwendet das mbed-Protokoll für das Flashen (alternativ waere "stlink" möglich)
upload_protocol = mbed
; Setzt die serielle Baudrate für den seriellen Monitor (Standard waere 9600)
monitor_speed = 115200
; Aktiviert die Unterstützung für Fließkommazahlen in printf()
build_flags = -Wl,-u_printf_float

lib_deps =  ; Definiert die Bibliotheken, die für das Projekt benötigt werden
  ; Treiber für I2C-LCD-Displays mit PCF8574 I/O-Expander
  mathertel/LiquidCrystal_PCF8574@^2.2.0
  ; Bibliothek für OLED SSD1306 I²C Display
  adafruit/Adafruit SSD1306@^2.5.12
  ; Scanner sür I²C Bus
  luisllamasbinaburo/I2CScanner@^1.0.1
```

Listing 1: Beispiel für die platformio.ini

Bibliotheken:[2][3][4]

[2] https://github.com/mathertel/LiquidCrystal_PCF8574?tab=readme-ov-file

[3] https://github.com/adafruit/Adafruit_SSD1306

[4] https://github.com/luisllamasbinaburo/Arduino-I2CScanner

1.5.8. README.md (optional)

Beschreibung: Eine optionale Datei, die typischerweise eine Beschreibung des Projekts, Anweisungen zur Installation und Verwendung, und andere relevante Informationen enthält.
Inhalt: Markup-Text (Markdown), der die Dokumentation für das Projekt bietet.
Hinweis: Diese Datei ist nützlich, um Informationen über das Projekt zu kommunizieren, insbesondere wenn das Projekt auf GitHub oder einer ähnlichen Plattform gehostet wird.

1.6. Erzeugtes Programmtemplate

PlatformIO erzeugt ein C++ Programmtemplate mit dem Namem main.cpp. Die entpricht sozusagen einem “HelloWorld. Die C++-Library des Arduino Framework muss hier mit `#include <Arduino.h >` eingebnunden werden.

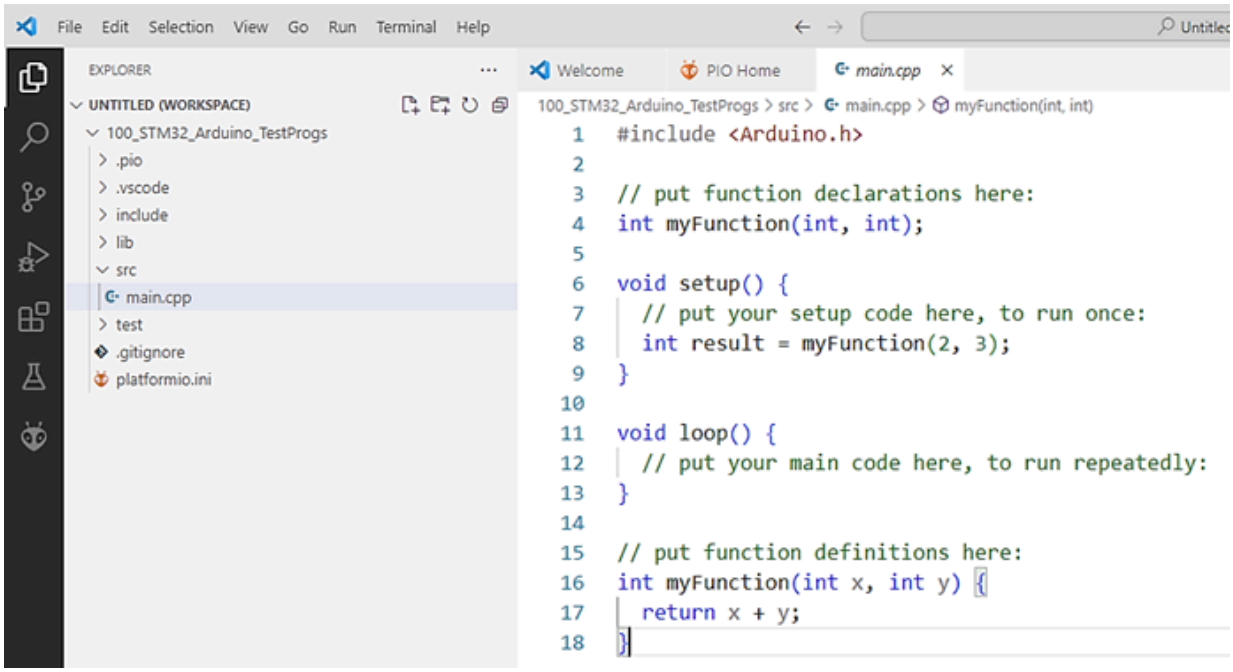

Abbildung 4: Erzeugtes C++ Tmeplate im Ordner src/

1.7. Effizientes Programmieren von Übungen mit Präpozessor Switches

Wenn zahlreiche unterschiedliche Übungen erstellt werden sollen, ist es effizienter, ein Projekt mit sogenannten Präprozessor-Switches zu erstellen. In einem solchen Projekt können mehrere C++ -Dateien enthalten sein, die jeweils eigene `void setup()`- und `void loop()`-Funktionen definieren.

Allerdings sollte zu einem Zeitpunkt immer nur eine Datei aktiv sein. Dies wird durch Präprozessor-Anweisungen gesteuert:

- Verwenden Sie `#if 1`, um eine Datei zu aktivieren.
- Verwenden Sie `#if 0`, um eine Datei zu deaktivieren.

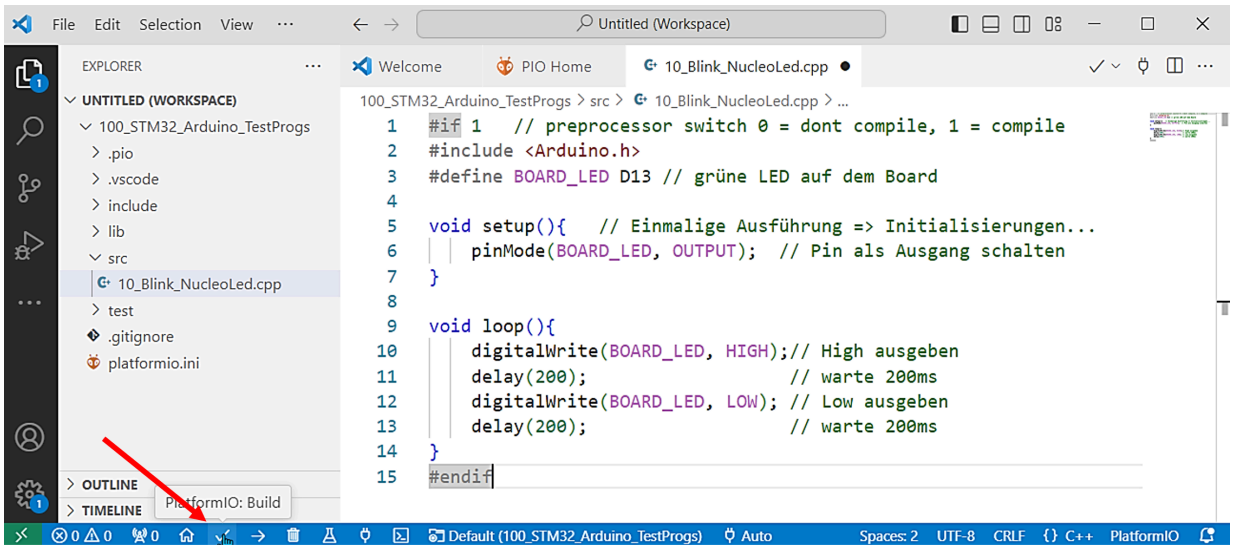

Abbildung 5: Erzeugtes C++ Template im Ordner src/

1.8. 100_Testprogramme_STM32_Arduino

An dieser Stelle werden grundlegende Einstiegsprogramme vorgestellt. Grundlage dafür ist die Sprachreferenz von Arduino, die in drei Kategorien unterteilt ist: Funktionen, Variablen und Struktur. Siehe **https://docs.arduino.cc/language-reference/de/**

1.8.1. 10_Blink_NucleoLed_LCD.cpp

Das Programm 10_Blink_NucleoLed_LCD.cpp steuert eine grüne LED auf einem Nucleo-Board (Pin PA5) an und gibt Text auf einem I2C-LCD-Display aus. Es verwendet die Bibliothek LiquidCrystal_PCF8574 zur Steuerung des LCDs. Die Hauptfunktionen des Programms sind:

- Initialisierung der grünen LED: Der Pin PA5 wird als Ausgang konfiguriert.
- LCD Initialisierung: Das LCD-Display wird über I2C mit den entsprechenden SCL- und SDA-Pins (PB8 und PB9) eingerichtet und die Helligkeit des Displays wird auf maximal gesetzt.
- Anzeige von Text: Es wird der Text “BLINK” in der ersten Zeile und “on PA5” in der zweiten Zeile des LCDs angezeigt.[5]
- Blinken der LED: Die LED wird alle 200 Millisekunden ein- und ausgeschaltet.

Das Programm läuft in einer Endlosschleife, wobei die LED blinkt und der Text konstant auf dem LCD angezeigt wird.

```cpp
// 10_Blink_NucleoLed_LCD.cpp
// Dieses Programm schaltet die grüne LED auf dem Nucleo Board an und aus
// und gibt den Text "BLINK" und "on PA5" auf einem LCD Display aus
// Das LCD Display wird über I2C angesteuert Das Programm verwendet die
// Bibliothek LiquidCrystal_PCF8574

// M. Schreger, 26.2.2025
// Version 1.0
// -----------------------------------------------------------

#if 1   // preprocessor switch 0 = don't compile, 1 = compile
#include <Arduino.h>
#include <LiquidCrystal_PCF8574.h>

#define BOARD_LED PA5 // grüne LED auf dem Nucleo Board

LiquidCrystal_PCF8574 lcd(0x27); // erzeuge das Objekt lcd und setze die I2C Adresse des LCDs

// Funktion zur Initialisierung des LCD
void initLcd() {
  Wire.setSCL(PB8);  // set I2C Clock Line
  Wire.setSDA(PB9);  // set I2C Data Line
  lcd.begin(16, 2); // setze die Anzahl der Zeichen und Zeilen des LCDs
  lcd.setBacklight(255); // setze die Helligkeit des LCDs
}

// Funktion zur Ausgabe von Text auf dem LCD
void lcdOut(String z1, String z2) {
  lcd.clear(); // lösche den Bildschirm
  lcd.setCursor(0, 0); // setze den Cursor auf die erste Zeile
  lcd.print(z1); // gebe den Text "Hello, World!" aus
  lcd.setCursor(0, 1); // setze den Cursor auf die zweite Zeile
  lcd.print(z2); // gebe den Text "LCD Display" aus
}

void setup() {   // Einmalige Ausführung => Initialisierungen...
  pinMode(BOARD_LED, OUTPUT);  // Pin als Ausgang schalten
  initLcd(); // initialisiere das LCD
  lcdOut("BLINK", "on PA5"); // gebe Text auf dem LCD aus
}

void loop(){
    digitalWrite(BOARD_LED, HIGH);// High ausgeben
    delay(200);                   // warte 200ms
    digitalWrite(BOARD_LED, LOW); // Low ausgeben
    delay(200);                   // warte 200ms
}
#endif
```

Listing 2: Blink Programm mit Lcd Ausgabe

1.8.2. 11_SerialPrintCounter.cpp

Das Programm initialisiert einen Zähler und gibt diesen in einer Endlosschleife zusammen mit dem Text “Count:” über die serielle Schnittstelle aus. Die serielle Kommunikation wird im Setup mit einer Baudrate von 112500 gestartet. In jeder Iteration der Loop-Funktion wird der Zähler

[5]https://docs.arduino.cc/language-reference/de/variablen/data-types/stringObject/

um 1 erhöht und anschließend der Text “Count:” gefolgt vom aktuellen Zählerstand und einem Zeilenumbruch ausgegeben.

```
#if 1   // preprocessor switch 0 = don't compile, 1 = compile
#include <Arduino.h>
uint32_t counter = 0; // Zähler initialisieren

void setup(){   // Einmalige Ausführung => Initialisierungen...
    Serial.begin(112500); // Baudrate
}

void loop(){
  Serial.print("Count:"); //  Text ausgeben
  counter++;   // Zähler erhöhen
  Serial.print(counter);  // Zähler ausgeben
  Serial.print("\r\n");  // Zeilenumbruch
}
#endif
```

Listing 3: Serielle Ausgabe mit 112500 Baud

1.8.3. 12_SerialPrintTime.cpp

Das Programm gibt die Zeit, die seit dem Start des Programms vergangen ist, in Sekunden über die serielle Schnittstelle aus. Dabei wird in jeder Schleife die Zeit in Millisekunden abgefragt, in Sekunden umgerechnet und ausgegeben. Das Programm startet die serielle Kommunikation und gibt in jeder Schleife die verstrichene Zeit seit Programmstart (in Sekunden) mit dem Präfix “Zeit: “ aus

```
#if 1   // preprocessor switch 0 = don't compile, 1 = compile
#include <Arduino.h>
uint32_t zeit = 0;
void setup(){   // Einmalige Ausführung => Initialisierungen...
    Serial.begin(112500);  // Ändern
}

void loop(){
  Serial.print("Zeit: ");
  zeit = millis();    // Zeit in ms seit Start des Programms
  Serial.print(zeit/1000.0); // Zeit in Sekunden
  Serial.print("\r\n");
}
#endif
```

Listing 4: Vergangene Zeit

1.8.4. 13_SerialPlotter.cpp

Unter Extentions impotrieren sie bitte den **serial-plotter**[6] von Mario Zechner. Beenden sie bitte den Seriellen Monitor (WICHTIG). Laden Sie folgendes Porgramm in den Controller.

[6] https://marketplace.visualstudio.com/items?itemName=badlogicgames.serial-plotter

```ino
#if 1   // preprocessor switch 0 = don't compile, 1 = compile
#include <Arduino.h>
// In VS Code, press CTRL + SHIFT + P
// A serial port can only be accessed by one process at a time. You cannot monitor
// a port and upload a program to the Arduino simultaneously. Ensure no other
// process is using the port if the plotter can't connect. Similarly, if you're
// having trouble uploading to the Arduino, make sure //the serial plotter
// isn't monitoring the port. If you create multiple plots with specific variable
// selections, they need to be recreated when starting a new session.
//
void setup() {
  Serial.begin(112500);
}

float angle = 0;
void loop() {
  Serial.print(">");

  Serial.print("var1:");
  Serial.print(cos(angle));
  Serial.print(",");

  Serial.print("var2:");
  Serial.print(cos(angle + PI / 2) * 0.1);
  Serial.print(",");

  Serial.print("var3:");
  Serial.print(cos(angle + PI / 4) * 1.2 + 2);
  Serial.println(); // Writes \r\n

  Serial.println("This is totally ignored");
  delay(100);

  angle += PI / 10;
}
#endif
```

Listing 5: Blink Programm mit Lcd Ausgabe

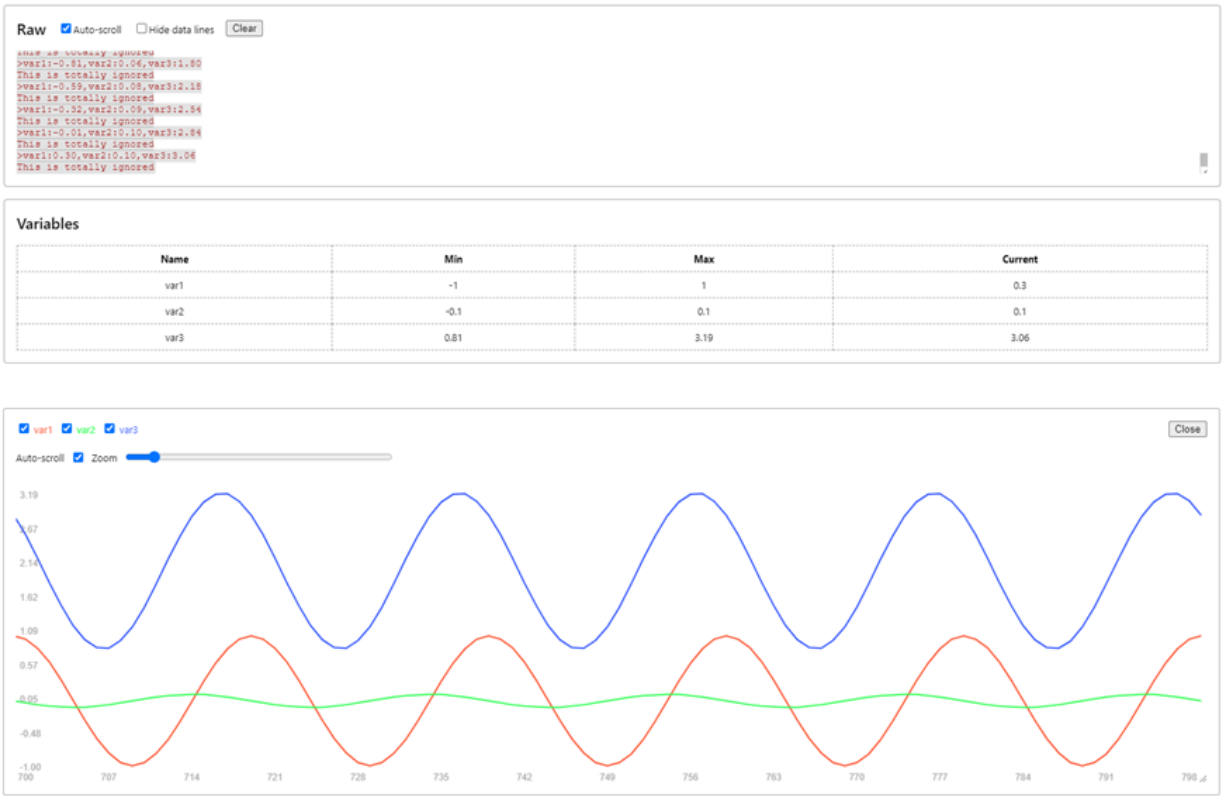

Abbildung 7: Ausgabe mit der Serial-Plotter Extension

1.8.5. 10_Blink_NucleoLed_LCD.cpp

```
#if 1
#include <Arduino.h>
#include <LiquidCrystal_PCF8574.h>
//SDA Pin PB9 - SCL Pin PB8

LiquidCrystal_PCF8574 lcd(0x27);  // set the LCD address to 0x27 for a 16 chars and 2 line display
int show = -1;

void setup() {
  int error;
  Serial.begin(9600);
  Serial.println("LCD...");
    // wait on Serial to be available
  while (!Serial);
  Serial.println("Probing for PCF8574 on address 0x27...");
  // See http://playground.arduino.cc/Main/I2cScanner how to test for a I2C device.
  Wire.setSCL(PB8);
  Wire.setSDA(PB9);
  Wire.begin();
  Wire.beginTransmission(0x27);
  error = Wire.endTransmission();
  Serial.print("Error: ");
  Serial.print(error);

  if (error == 0) {
    Serial.println(": LCD found.");
    show = 0;
    lcd.begin(16, 2);  // initialize the lcd
  } else {
    Serial.println(": LCD not found.");
  }
}
void loop() {
  if (show == 0) {
    lcd.setBacklight(255);
    lcd.home();
    lcd.clear();
    lcd.print("Hello LCD");
    delay(1000);
    lcd.setBacklight(0);
    delay(400);
    lcd.setBacklight(255);

  } else if (show == 1) {
    lcd.clear();
    lcd.print("Cursor On");
    lcd.cursor();
  } else if (show == 2) {
    lcd.clear();
    lcd.print("Cursor Blink");
    lcd.blink();
  } else if (show == 3) {
    lcd.clear();
    lcd.print("Cursor OFF");
    lcd.noBlink();
    lcd.noCursor();
  } else if (show == 4) {
    lcd.clear();
    lcd.print("Display Off");
    lcd.noDisplay();
```

```
  } else if (show == 5) {
    lcd.clear();
    lcd.print("Display On");
    lcd.display();
  } else if (show == 7) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("*** first line.");
    lcd.setCursor(0, 1);
    lcd.print("*** second line.");
  } else if (show == 8) {
    lcd.scrollDisplayLeft();
  } else if (show == 9) {
    lcd.scrollDisplayLeft();
  } else if (show == 10) {
    lcd.scrollDisplayLeft();
  } else if (show == 11) {
    lcd.scrollDisplayRight();
  } else if (show == 12) {
    lcd.clear();
    lcd.print("write-");
  } else if (show == 13) {
    lcd.clear();
    lcd.print("custom 1:<\01>");
    lcd.setCursor(0, 1);
    lcd.print("custom 2:<\02>");
  } else {
    lcd.print(show - 13);
  }  // if
  delay(1400);
  show = (show + 1) % 16;
}  // loop()
#endif
```

Listing 7: Blink Programm mit Lcd Ausgabe