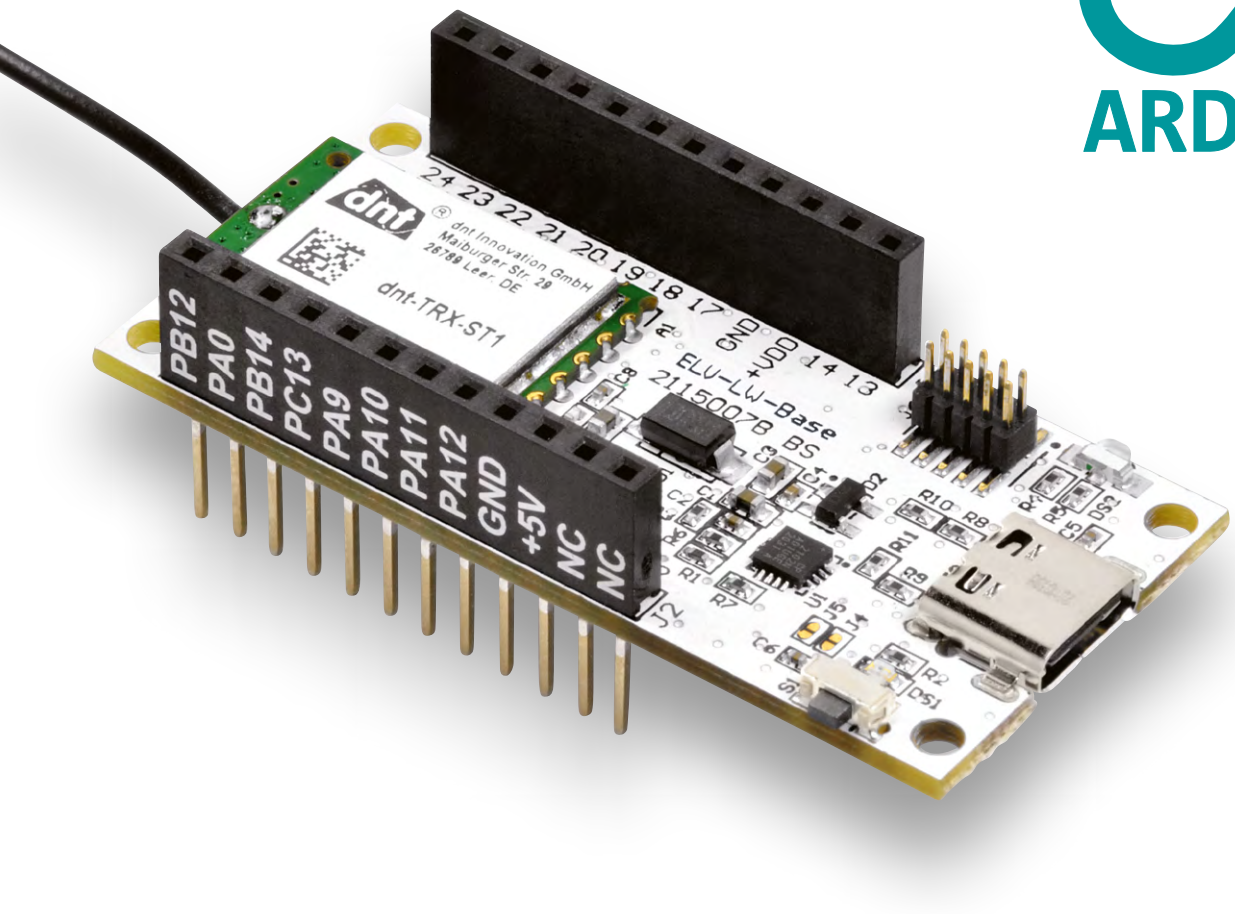
ELV-Modulsystem + Arduino – die Verschmelzung zweier Welten

Die Verschmelzung zweier Welten war immer schon begeisternd. Sie ist ein Nährboden für neue Ideen und eine treibende Kraft in Bezug auf kreatives Denken. Durch die Kombination vordefinierter Elektronikbausteine lassen sich neue Geräte- bzw. Produktideen auf schnellstem Wege realisieren. Man nehme ELV Basismodul A, füge ELV Applikationsmodul B hinzu, kombiniere dies mit der Arduino IDE und versorge alles mit intelligentem Programmcode. Et voilà, ein lang gehegter Gerätewunsch erfüllt sich. Durch die Arduino IDE, die ELV Basis- und Applikationsmodule sowie die online verfügbaren Programmbeispiele und -bibliotheken wird man in die Lage versetzt, den Entwicklungsaufwand eines Geräts deutlich zu vereinfachen und zugehörige Entwicklungszeiten drastisch zu verkürzen. So sind der Vorstellungskraft keine Grenzen gesetzt.



Motivation

Mit der Integration des ELV-Modulsystems in die Arduino-Umgebung werden vielfältige Möglichkeiten geschaffen, um das ELV-Modulsystem auf persönliche Bedürfnisse zuzuschneiden. Dieser Artikel soll eine Installationsanleitung sein sowie einige Codebeispiele und Denkanstöße geben, um eine Applikation unter Zuhilfenahme der Arduino IDE zu entwerfen.

Anleitung

Um die ELV LoRaWAN®-Base mit der Arduino IDE verwenden zu können, muss zunächst das STM32duino-Paket in der Arduino-Entwicklungsumgebung installiert werden. Hierzu navigiert man im Menüband der Arduino-IDE-Applikation zunächst auf *File* und dann zu *Preferences...*, um in das Einstellungsfenster zu gelangen (Bild 1).

Innerhalb der Einstellungen befindet sich am unteren Ende des Fensters ein Feld mit der Beschriftung *Additional Board Manager URLs*. An dieser Stelle muss die URL eingetragen werden, damit das STM32duino-Paket installiert werden kann (Bild 2). Bei Bestätigung mit OK wird die Option der Boards in den Arduino-Board-Manager eingetragen (Bild 3).

Bei den Board-Definitionen handelt es sich um eine Auflistung verschiedener Microcontroller-Platinen und deren Eigenschaften. Einige Hersteller haben z. B. aus Platzgründen nicht alle Anschlusspins des verwendeten Microcontrollers nach außen zugänglich gemacht oder über bestimmte Beschaltungselektronik einige Funktionen von Pins bevorzugt bzw. unterbunden. Mit der Auflistung dieser Elemente wird somit gewährleistet, dass jeder Eintrag auch mit dem verwendeten Code funktionieren kann.

Um die Board-Definitionen tatsächlich herunterzuladen, muss in der Seitenleiste der Entwicklungsumgebung auf das Platinensymbol geklickt werden, um den Board-Manager zu öffnen. Hierzu wird in der Suchleiste der Suchbegriff „STM32 MCU based boards" eingegeben. Beim entsprechenden Eintrag kann im Drop-down-Menü der Versionen mindestens die Version 2.8.1 ausgewählt werden.

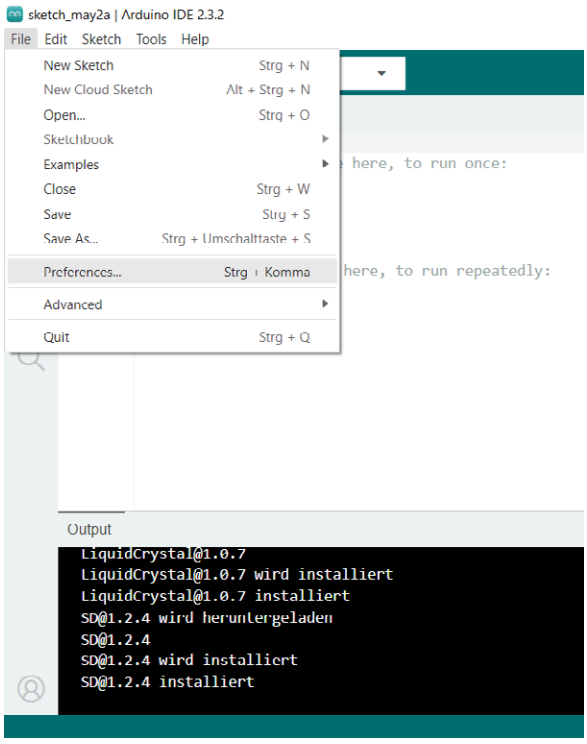


Bild 1: Das Einstellungsmenü ist im Menüband unter „File" zu finden.

Hinweis:
Um die Bibliothek verwenden zu können wird zusätzlich noch das Programm STM32CubeProgrammer benötigt.

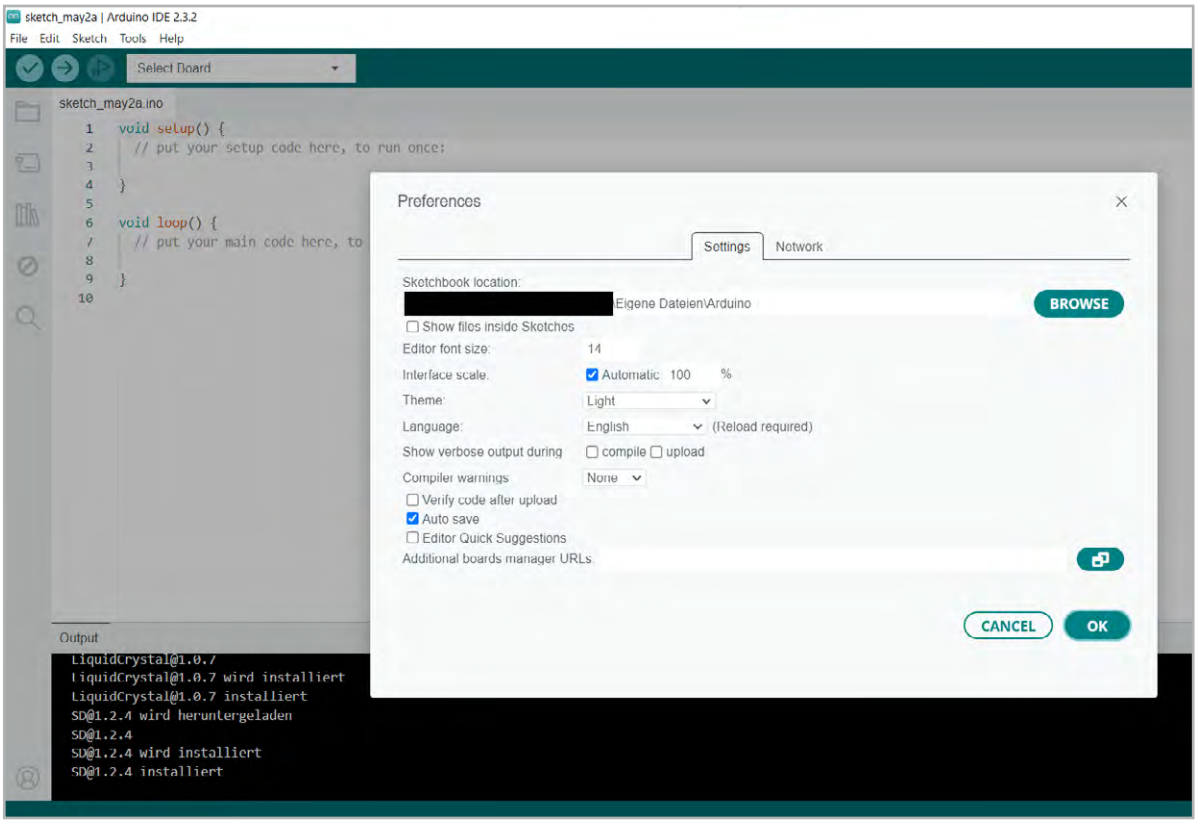


Bild 2: Im Feld „Additional boards manager URLs" muss die Adresse eingegeben werden.

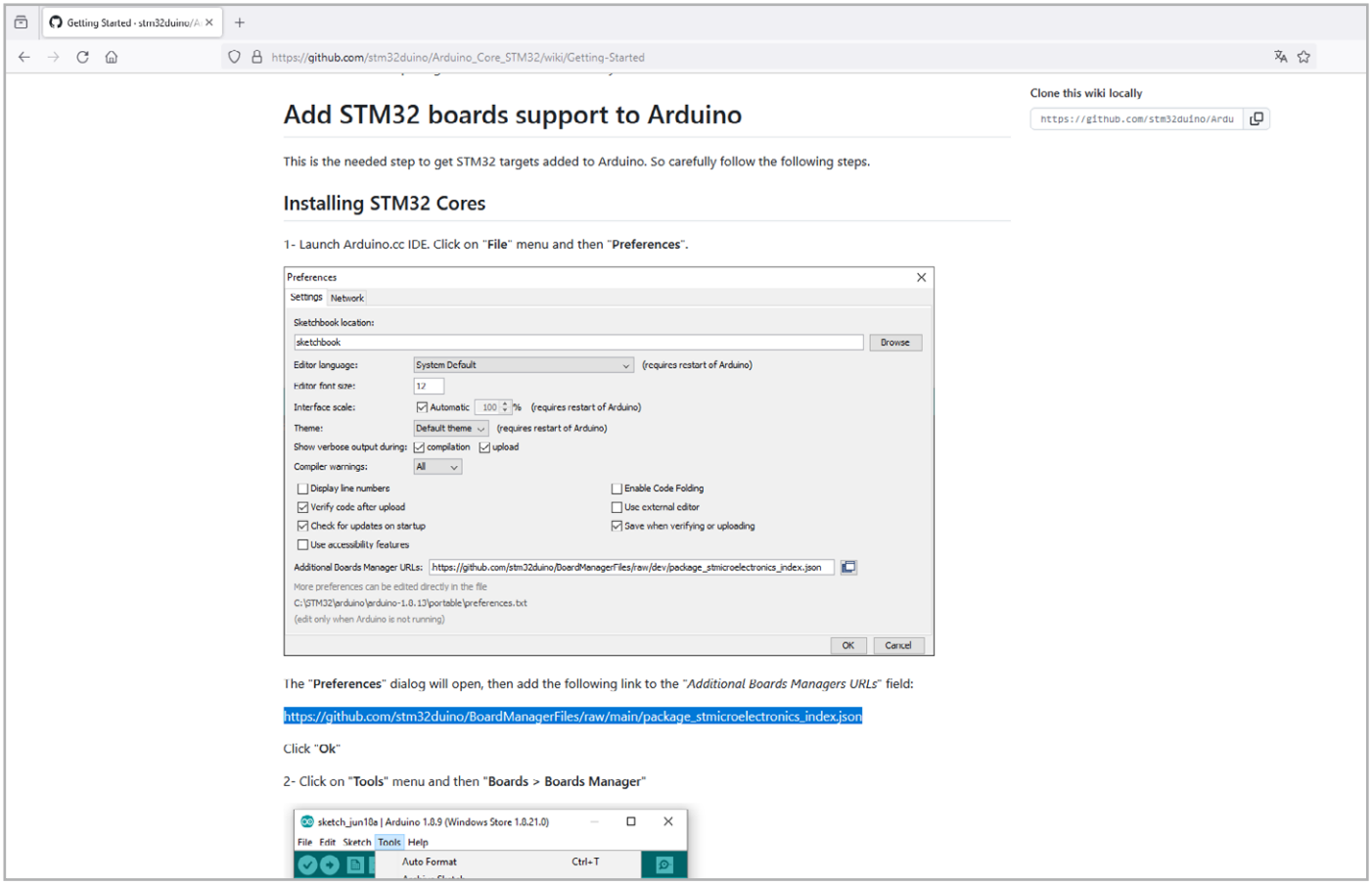


Bild 3: Die Board-Manager-Liste unter dieser URL-Adresse wird von STMicroelectronics stets aktuell gehalten.

Mit einem Klick auf *Install* werden die Board-Definitionen in die Entwicklungsumgebung übernommen (Bild 4).

Nach der erfolgreichen Installation kann die ELV LoRaWAN®-Base über das Menü ausgewählt werden. Dies geschieht wie in Bild 5 dargestellt über *Tools → Board → STM32 MCU based boards → ELV Modular System boards*.

Wenn das System ausgewählt wurde, kann unter *Tools → Board part number* die ELV LoRaWAN®-Base ausgewählt werden. Spätestens jetzt sollte sie auch per USB-C Kabel angeschlossen werden. Nun kann im Windows-Gerätemanager der COM-Port der Base ermittelt werden, wie in Bild 6 zu sehen ist. Dieser Port wird nachfolgend in der Arduino IDE angegeben.

Zuletzt muss unter *Upload method* der Eintrag *SMT32CubeProgrammer (Serial) with Bootloader* angewählt werden.

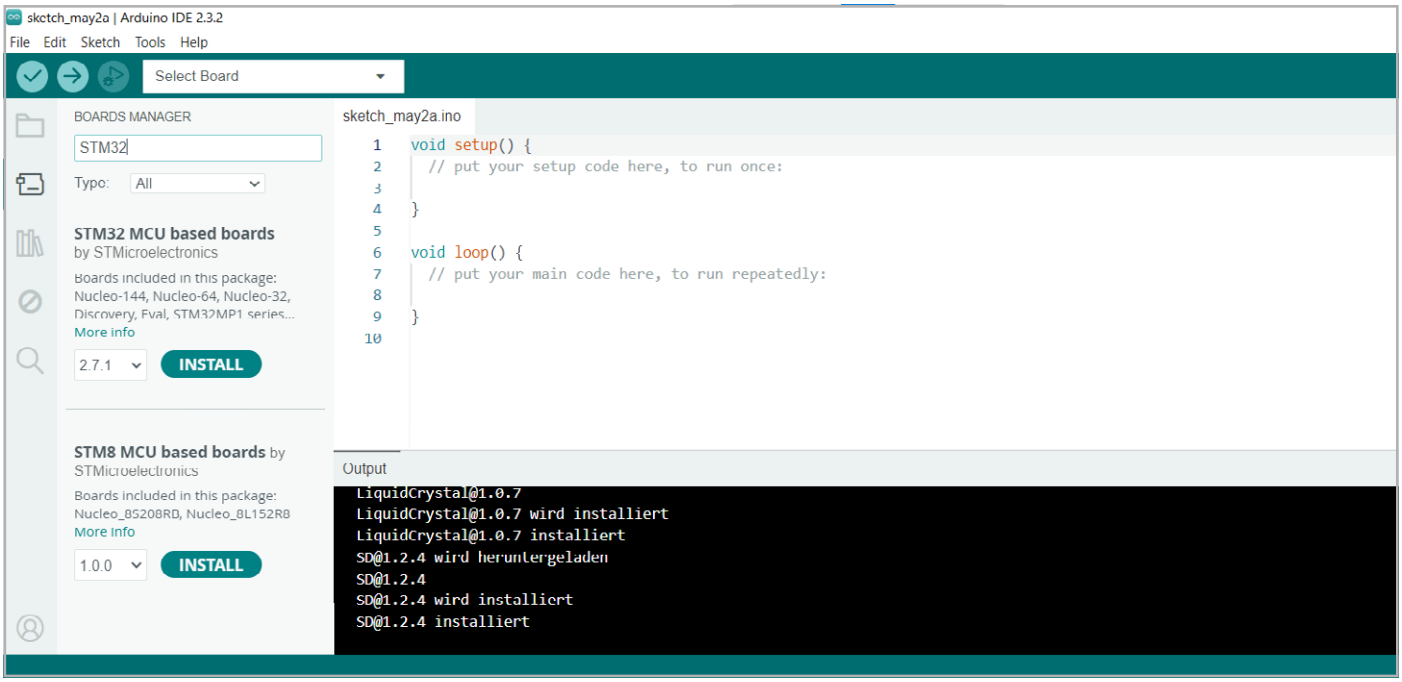


Bild 4: Nach erfolgreicher Installation sind die STM32-Board-Definitionen im Board-Manager der Arduino IDE zu finden.

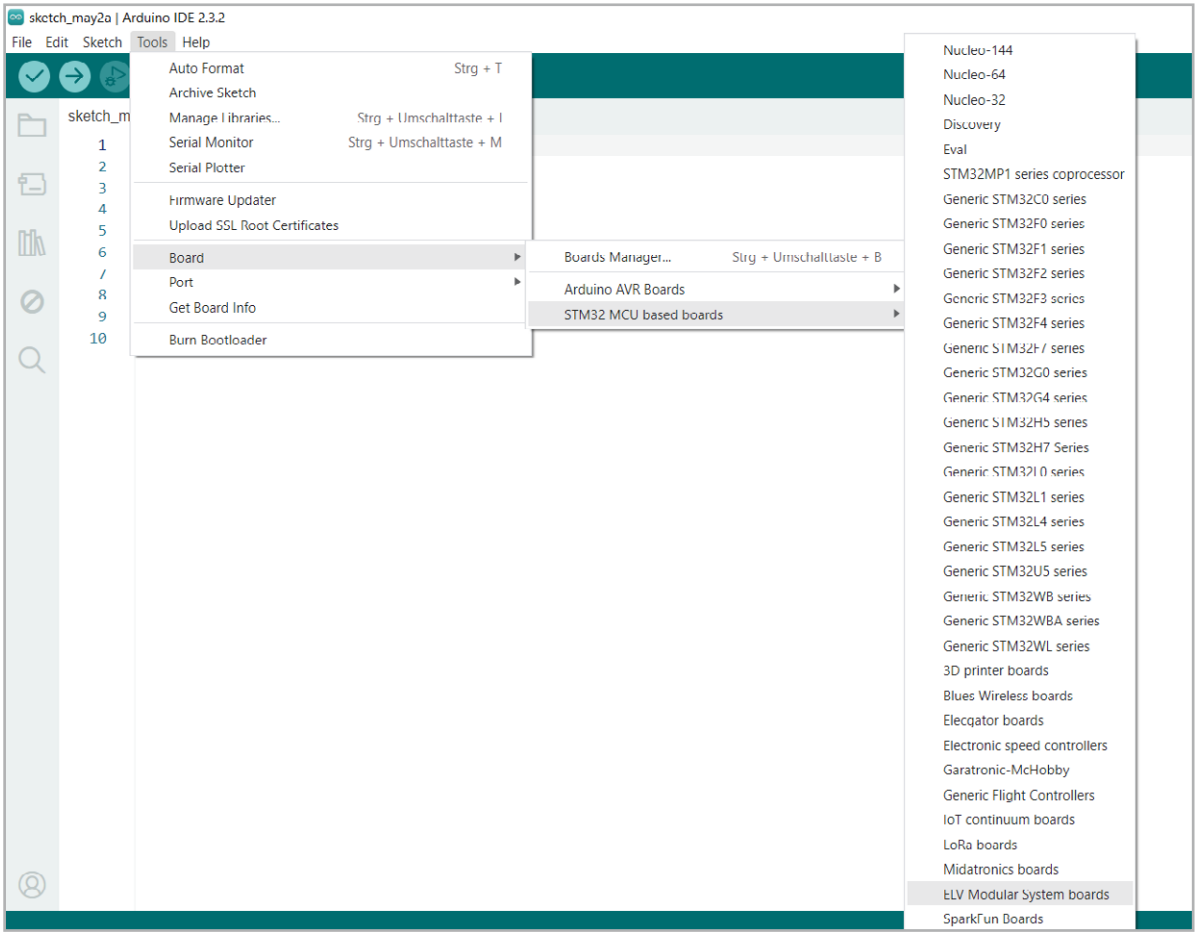


Bild 5: Nach erfolgreicher Installation kann nun der Eintrag der „ELV-modular-system"-Boards im Board-Menü gefunden werden.

Steht ein STLink-Programmer zur Verfügung, kann der Eintrag *SMT32CubeProgrammer (SWD) with Bootloader* verwendet werden. Hierfür muss dann SWDIO vom STLink mit Pin 18 und SWDCLK mit Pin 17 sowie GND mit Pin 16 und VDD mit Pin 15 verbunden werden.

Die korrekten Einstellungen zum Bespielen der ELV LoRaWAN®-Base sind in Bild 7 zusammenfassend dargestellt.

Nach dieser Einrichtung kann der erste Programmcode geschrieben werden, um die Installation zu prüfen. Hierzu kann das Blink.ino-Beispiel mit minimaler Änderung verwendet werden.

Um einen leichten Einstieg zu vollziehen, bietet es sich an, zunächst eine LED zum Blinken zu bringen. Das zugehörige Programm ist verhältnismäßig einfach. Die LED ist an einem bestimmten Pin angeschlossen. Dieser Pin muss zunächst als Ausgang initialisiert werden. Danach muss der Pin zwischen dem Logiklevel „HIGH" und „LOW" wechseln. Damit der Einschaltvorgang für das menschliche Auge sichtbar wird, muss die Einschaltdauer auf einen wahrnehmbaren Zeitwert eingestellt werden.

Das Beispielprogramm ist im Menü unter *File → Examples → 01. Basics → Blink* zu finden. Die Anpassungen im Programmcode sind im Bild Codeblock 1 zu finden.

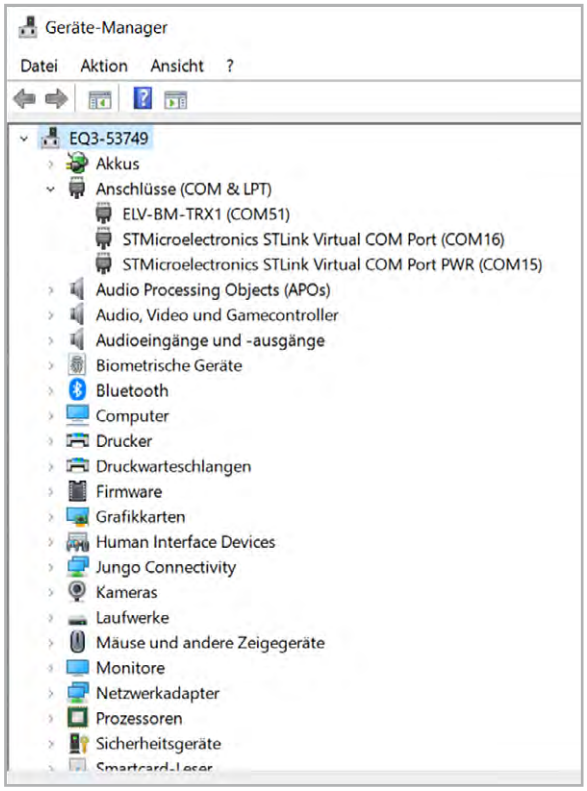


Bild 6: Der COM-Port kann im Windows-Gerätemanager nachgeschaut werden.

Alle Programmcodes finden Sie auch als Source-Datei im ELVshop bei diesem Fachbeitrag unter Downloads.

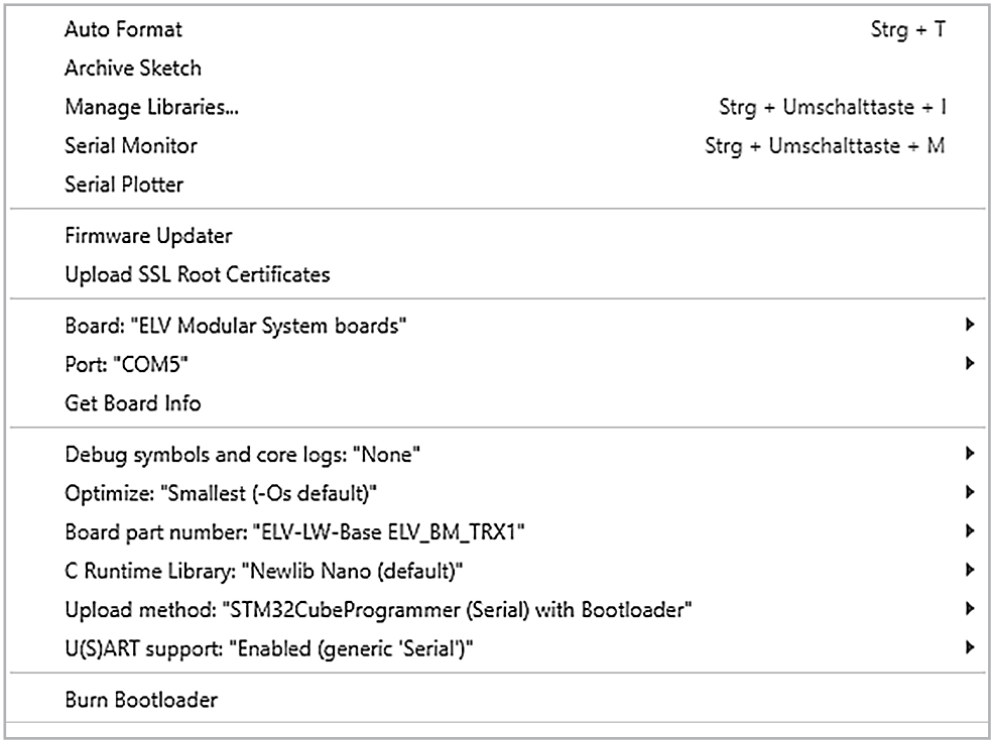


Bild 7: Die korrekten Programmiereinstellungen

Codeblock 1

```
// die setup-Funktion wird einmal zu Beginn ausgeführt
void setup() {
  // Die Board-LED der ELV-LoRaWAN® Base wird initialisiert (Grüne LED der
DUO-LED DS2
  pinMode(LED_BUILTIN, OUTPUT);
}

// Die loop()-Funktion wird immer wieder ausgeführt
void loop() {
  digitalWrite(LED_BUILTIN, LOW);  // LED einschalten (LOW ist das Logiklevel)
  delay(1000);                     // 1000 Millisekunden warten
  digitalWrite(LED_BUILTIN, HIGH); // LED wieder ausschalten, in dem die
Spannung auf HIGH gezogen wird
  delay(1000);                     // 1000 Millisekunden warten
}
```

Bild 8: Die Schaltflächen zum Verifizieren, Übertragen und Debuggen der Firmware.

Ob der Programmcode Syntaxfehler hat, kann mithilfe des Verify-Buttons anhand des Häkchen-Symbols geprüft werden (Bild 8). Dieser gibt bei erfolgreicher Kompilierung über die Konsole im unteren Fensterbereich Auskunft darüber, wie viel Speicherplatz das aktuelle Programm benötigt (Bild 9).

Flashen

Bevor der Programmcode auf die LoRaWAN®-Base übertragen werden kann, muss diese in den Bootloader-Modus gebracht werden. Dies geschieht, indem man beim Zuführen der Spannungsversorgung die Taste S1 gedrückt hält. Wenn der Bootloader aktiv ist, blinkt die rote LED an DS2 für eine Sekunde auf. Um den Flash-Vorgang zu starten, muss auf den Button mit dem Pfeil nach rechts (Bild 8) gedrückt werden. Nach dem erfolgreichen Flashen des Geräts beginnt die ELV LoRaWAN®-Base mit der Ausführung des Programms (Bild 9). Die grüne LED an DS2 sollte nun im Sekundentakt blinken gemäß den im Codeblock 1 definierten Vorgaben.

Verwendung von Spezialfunktionen

Eine Besonderheit des auf der ELV LoRaWAN®-Base verwendeten Funkmoduls ist der sehr energiesparende Low-Power-Modus. Um die darin enthaltenen Funktionen verwenden zu können, muss die Entwicklungsumgebung um eine Bibliothek erweitert werden. Eine solche Bibliothek stellt eine Sammlung von Programmcodes dar, die über programminterne Funktionen abrufbar ist. Die Bibliothek, die die Low-Power-Funktionen verwendbar macht, heißt STM32LowPower. Hier ist in der Beschreibung gesondert darauf zu achten, dass eine Abhängigkeit zu einer anderen Bibliothek besteht. Um die Low-Power-Funktionen verwenden zu können, muss die Real-Time-Clock des Controllers verwendet werden, da die anderen Timer im Low-Power-Modus abgeschaltet werden. Die Bibliothek, die diese Funktionalität bereitstellt, heißt STM32RTC. Das Installieren einer Bibliothek ist immer identisch und wird im Folgenden beschrieben.

Auf der Hauptseite der Bibliothek kann im rechten oberen Bereich der grüne „<> Code"-Button (siehe Bild 10) geklickt werden, wodurch sich ein Drop-down-Menü öffnet, das die Option *Download ZIP* verfügbar macht (Bild 11).

```
Sketch uses 12832 bytes (9%) of program storage space. Maximum is 129024 bytes.
Global variables use 1228 bytes (2%) of dynamic memory, leaving 47924 bytes for local variables. Maximum is 49152 bytes.
| | | ------------------------------------------------------------
| | | | | | | | | | |STM32CubeProgrammer v2.14.0
| | | ------------------------------------------------------------

Serial Port COM5 is successfully opened.
Port configuration: parity = even, baudrate = 115200, data-bit = 8,
| | | | | | | | | | |stop-bit = 1.0, flow-control = off
Activating device: OK
Board       : --
Chip ID: 0x497
BootLoader protocol version: 3.3
Device name : STM32WLxx
Flash size  : 256 KBytes (default)
Device type : MCU
Revision ID : --
Device CPU  : Cortex-M4


Memory Programming ...
Opening and parsing file: Blink.ino.bin
|  File          : Blink.ino.bin
|  Size          : 12.86 KB
|  Address       : 0x08000800


Erasing memory corresponding to segment 0:
Erasing internal memory sectors [1 7]
Download in Progress:


File download complete
Time elapsed during download operation: 00:00:02.014

RUNNING Program ...
|  Address:      : 0x8000800
Start operation achieved successfully
```

Bild 9: Während der Programmübertragung wird man von der Arduino IDE darüber informiert, welcher Prozess in Bearbeitung ist.

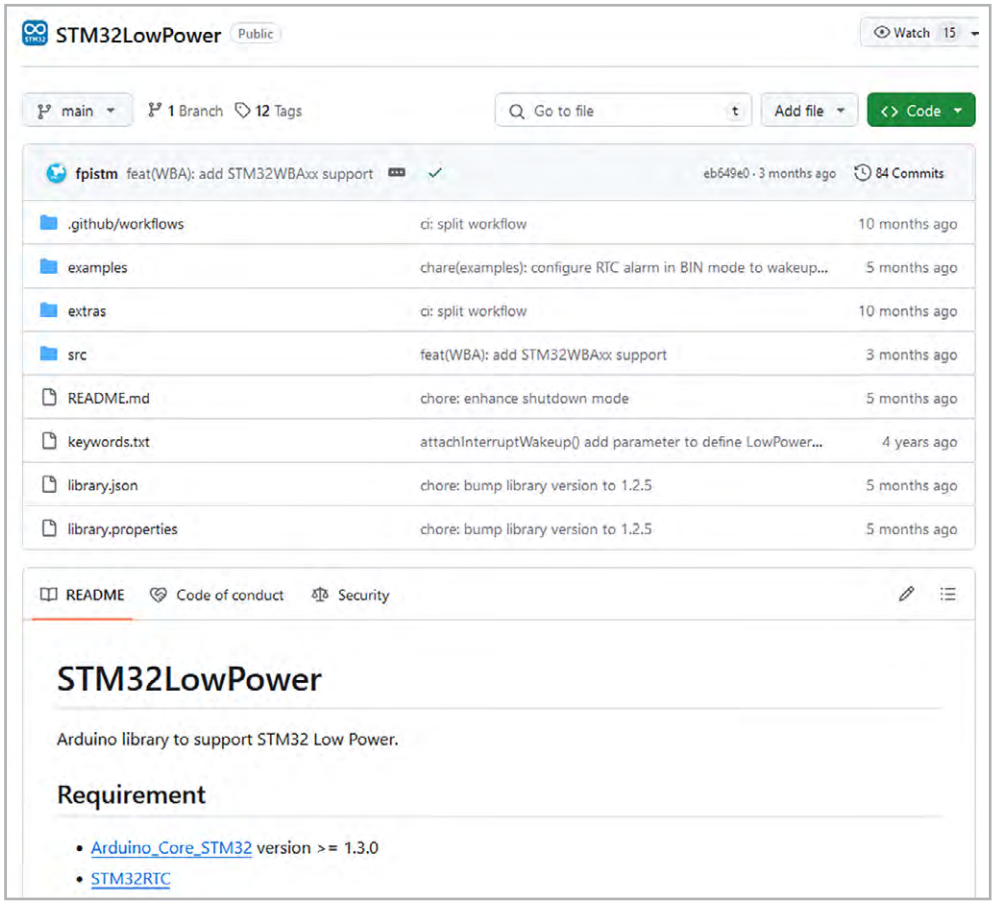


Bild 10: Alle Bibliotheken, die für das Programmieren der ELV LoRaWAN®-Base notwendig sind, sind im github von STMicroelectronics zu finden.

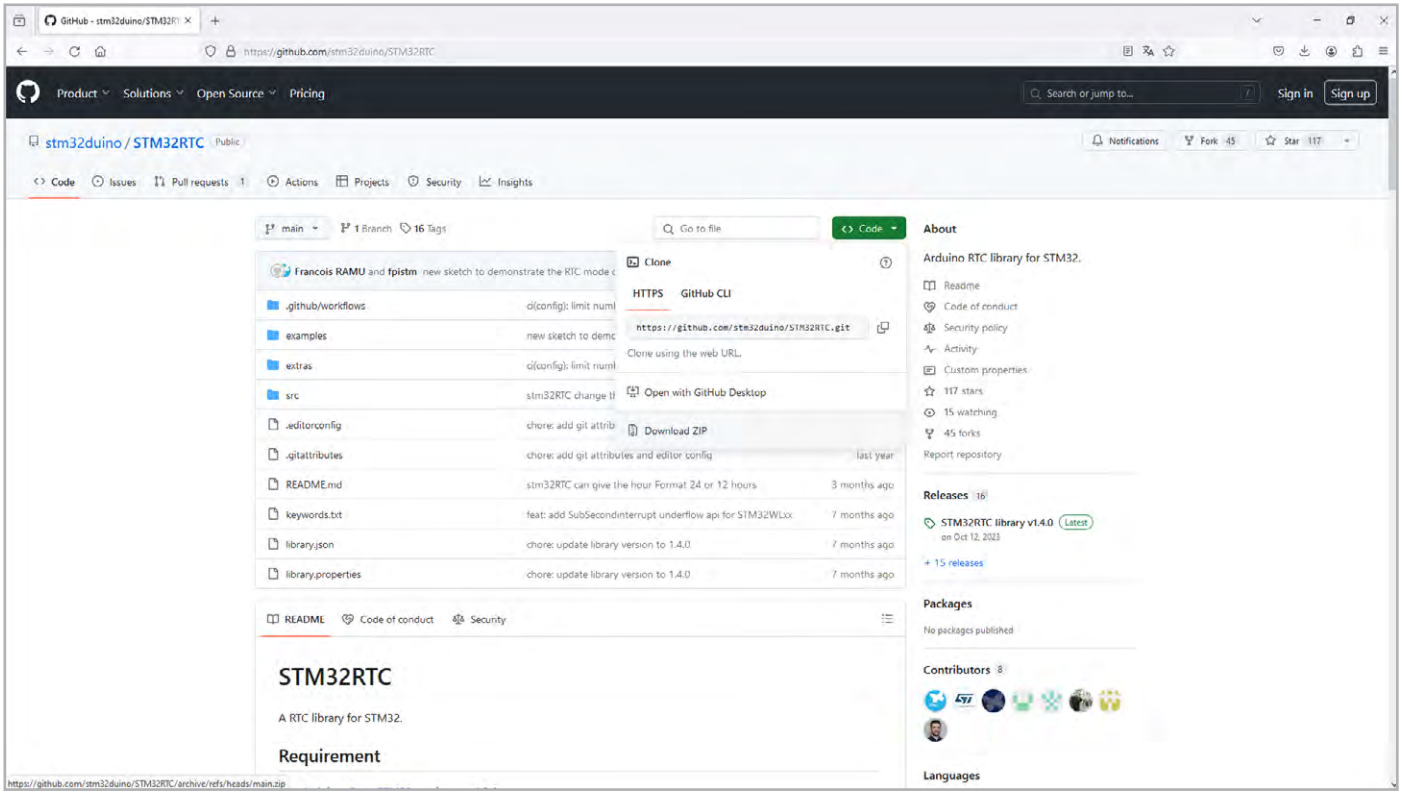


Bild 11: Um externe Programmbibliotheken verwenden zu können, muss als erster Schritt der Programmcode als ZIP-Archiv von github heruntergeladen werden.

Um die ausgewählte Bibliothek innerhalb der Arduino IDE verwenden zu können, muss im Menü die Option *Sketch → Include Library → Add .ZIP Library* gewählt werden. Dadurch öffnet sich ein Dateiauswahldialog, in dem die heruntergeladenen ZIP-Archive ausgewählt werden können (Bild 12). Sobald die STM32RTC und die STM32LowPower hinzugefügt worden sind, kann ein erneutes Programmbeispiel verwendet werden.

Das Programmbeispiel Codeblock 2 ist unter *File → Examples → STM32duino Low Power → TimedWakeup* zu finden. Die erforderlichen Anpassungen sind ähnlich zu denen im Beispiel Codeblock 1.

Nach dem Kompilieren und Hochladen des Programms ändert sich nichts am sichtbaren Verhalten des Geräts. Betrachtet man allerdings den Stromverbrauch, ist erkennbar, dass in den Deep-Sleep-Phasen nur noch ein Strom von 2,5 µA fließt. Im Vergleich zu den 3,4 mA im Normalmodus ist dies eine deutliche Verbesserung in etwa um den Faktor 1000.

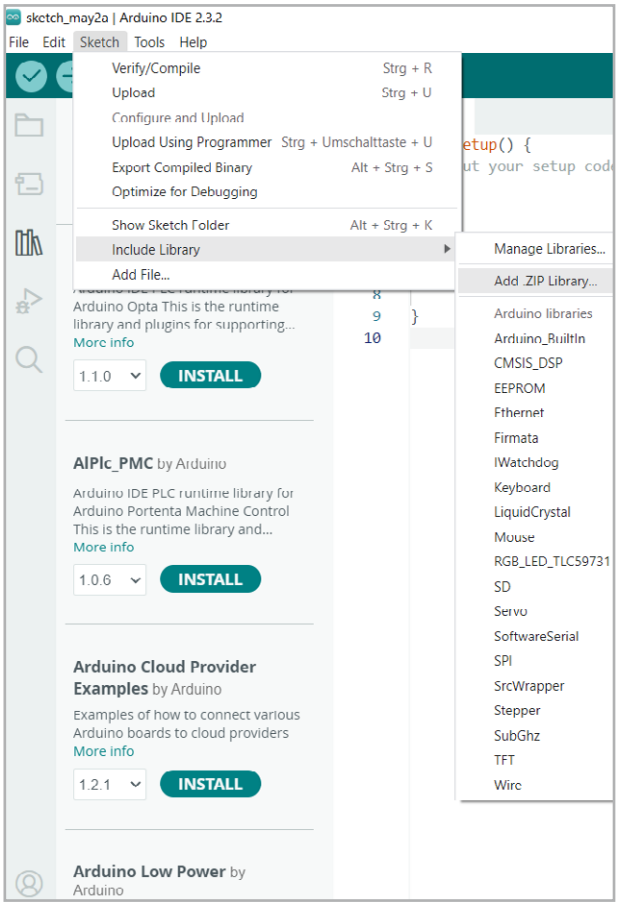


Bild 12: Der zweite Schritt zur Verwendung externer Bibliotheken ist das Einfügen in die Arduino-Entwicklungsumgebung.

Codeblock 2

```
/*
  TimedWakeup

  This sketch demonstrates the usage of Internal Interrupts to wakeup a chip
  in deep sleep mode.

  In this sketch, the internal RTC will wake up the processor every second.

  This example code is in the public domain.
*/
// Die Verwendete Bibliothek muss über eine include-Dirketive zum Code
hinzugefügt werden

#include "STM32LowPower.h"

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  // Die LowPower-Funktionen müssen einmal initialisiert werden
  LowPower.begin();
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  LowPower.deepSleep(1000);
  digitalWrite(LED_BUILTIN, LOW);
  LowPower.deepSleep(1000);
}
```

Codeblock 3

```
//Die Bibliothek wird über die include-Direktive eingeführt
#include <STM32LoRaWAN.h>

STM32LoRaWAN modem; //ein Objekt für die Verwendung der Bibliothek

static const unsigned long TX_INTERVAL = 60000; /* ms */
unsigned long last_tx = 0;

/* Get the rtc object */
STM32RTC& rtc = STM32RTC::getInstance(); //Ein Objekt für die Realtime Clock instantiiert

void setup() {
  Serial.begin(115200);
  Serial.println("Start");
  modem.begin(EU868);

  // Hier wird die Join-Methode ausgewählt, die Verwendung mit TTN benötigt die OTAA (Over the Air Authentification) . Die Gruppen von Nullen wird ersetzt durch die JoinEUI, AppKey und DevEUI der LoRaWAN Base. Diese können über den mitgelieferten QR-Code Sticker oder das ELV Flasher Tool ermittelt werden
  bool connected = modem.joinOTAA(/* JoinEui */ "0000000000000000", /* AppKey */ "00000000000000000000000000000000", /* DevEui */ "0000000000000000");
  //bool connected = modem.joinABP(/* DevAddr */ "00000000", /* NwkSKey */ "00000000000000000000000000000000", /* AppSKey */ "00000000000000000000000000000000");

  if (connected) {
    Serial.println("Joined");
  } else {
    Serial.println("Join failed");
    while (true) /* Endlosschleife */
      ;
  } //Wenn die Verbindung erfolgreich war, wird das Programm fortgeführt, wenn es nicht erfolgreich war, bleibt es stehen

  /* set the calendar */
  rtc.setTime(00, 00, 00); //Hier kann die aktuelle Uhrzeit eingetragen werden
  rtc.setDate(04, 07, 23); //Hier kann das aktuelle Datum angegeben werden
}

void send_packet() {
  char payload[27] = { 0 }; /* Die Länge des Pakets */
  /* Hier wird das Paket erstellt - mit dem Datum und der Uhrzeit */
  sprintf(payload, "%02d/%02d/%04d - %02d:%02d:%02d",
          rtc.getMonth(), rtc.getDay(), 2000 + rtc.getYear(),
          rtc.getHours(), rtc.getMinutes(), rtc.getSeconds());
  modem.setPort(10);
  modem.beginPacket();
  modem.write(payload, strlen(payload));

  if (modem.endPacket() == (int)strlen(payload)) {
    Serial.println("Sent packet");
  } else {
    Serial.println("Failed to send packet");
  }
      //Nach Senden des Pakets wird überprüft, ob ein Downlinkpaket empfangen wurde
  if (modem.available()) {
    Serial.print("Received packet on port ");
    Serial.print(modem.getDownlinkPort());
    Serial.print(":");
    while (modem.available()) {
      uint8_t b = modem.read();
      Serial.print(" ");
      Serial.print(b >> 4, HEX);
      Serial.print(b & 0xF, HEX);
    }
    Serial.println();
  }
}

void loop() {
      //Wenn das letzte Paket länger her ist, als der TX_INTERVAL, dann schicke ein neues Paket
  if (!last_tx || millis() - last_tx > TX_INTERVAL) {
    send_packet();
    last_tx = millis();
  }
}
```

LoRaWAN®

Das wichtigste Herausstellungsmerkmal der ELV LoRaWAN®-Base ist die LoRaWAN®-Funktionalität. Um diese verwenden zu können, wird die Bibliothek STM32LoRaWAN benötigt. Diese ist ebenfalls von der STM32RTC-Bibliothek abhängig, die bereits im vorherigen Schritt installiert wurde.

Nach der Installation der STM32LoRaWAN®-Bibliothek kann das Programmbeispiel Codeblock 3 verwendet werden, um die LoRaWAN®-Funktion überprüfen zu können. Hierbei ist darauf zu achten, anstelle der Nullen die Keys der ELV LoRaWAN®-Base einzusetzen.

Die ELV LoRaWAN®-Base kann nun wie im Fachbeitrag zur ELV-LW-Base beschrieben mithilfe der gleichen Keys registriert werden. Um die Daten innerhalb des TTN-Netzwerks auslesen zu können, muss dort in der Oberfläche ein Payload-Parser definiert werden. Der Code für das Umwandeln von Hexadezimalwerten zurück in ASCII-Zeichen ist in Codeblock 4 dargestellt. Dieser Code wird im TTN als Uplink-Payload-Parser entweder für die gesamte Applikation oder das einzelne Endgerät eingefügt.

Codeblock 4

```
function Decoder(bytes, port) {
  // Decode plain text; for testing only
  return {
      myTestValue: String.fromCharCode.apply(null, bytes)
  };
}
```

Verwendung mit Applikationsmodulen des ELV-Modulsystems

Nachfolgend wird das Vorgehen zur Verwendung verschiedener ELV-Applikationsmodule erklärt, beispielhaft anhand des ELV-Applikationsmoduls Luftdruck ELV-AM-AP. An dieser Stelle erfolgt allerdings keine Schritt-für-Schritt-Anleitung, sondern lediglich ein Überblick, um ein grundlegendes Gefühl für das notwendige Vorgehen zu vermitteln. Dazu wird im Library-Manger der Arduino-Entwicklungsumgebung nach der Teilenummer des auf dem ELV-Applikationsmodul verbauten Sensors gesucht. Für das Beispiel des ELV-AM-AP ist dies der Sensor BMP581 von Bosch Sensortec. Um den Library-Manager zu verwenden, muss in der Seitenleiste der Arduino IDE auf das Bibliotheksmanager-Icon geklickt werden. In der Suchleiste kann anschließend nach der Bezeichnung des Sensors gesucht werden, wie in Bild 13 zu sehen.

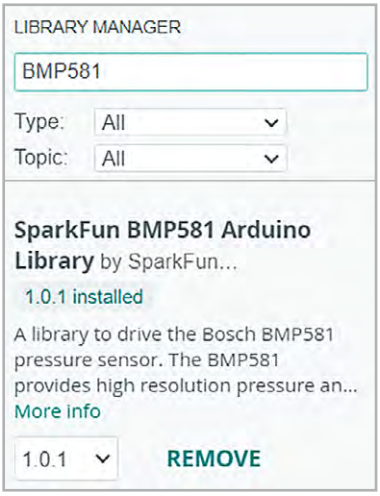


Bild 13: Die für das Modul ELV-AM-AP benötigte Bibliothek

Nach der Installation dieser Bibliothek stehen Programmbeispiele des BMP581 zur weiteren Verfügung. Unter *File → Example → Sparkfun BMP581 Arduino Library → Example1_BasicReadingsI2C* kann man sich einen Überblick darüber verschaffen, wie diese Bibliothek arbeitet. Von besonderer Bedeutung sind hierbei die #include-Direktiven, Initialisierungen sowie das Auslesen der Messwerte. Verdeutlicht wird dies in Codeblock 5.

Alle Codeblocks dieses Betrags können zu einem neuen Programm zusammengeführt werden. Dies wird in Codeblock 6 veranschaulicht.

Zudem ergeben sich durch die Integration der ELV LoRaWAN®-Base einige weitere Möglichkeiten, um das ELV-Modulsystem in spannenden Projekten einzusetzen. Es können mehrere ELV-Applikationsmodule zusammengesteckt verwendet werden, wodurch der Kreativität unserer Leser kaum Grenzen gesetzt sind.

Ein interessantes Beispiel ist die Realisation einer eigenen Schaltaktorik, die man mithilfe eines Step-up-Spannungswandlers, des ELV Bausatzes Relais-Schaltmodul ELV-RSM1 und eines zum Verbraucher passenden Relais erstellt.

Neben der Einbindung von ELV-Applikationsmodulen können auch viele individuelle Schaltungen mithilfe der ELV Prototypenadapter entwickelt werden. Hierzu wird es in den kommenden ELVjournalen weitere spannende Projekte geben.

Codeblock 5

```
#include <Wire.h>
#include "SparkFun_BMP581_Arduino_Library.h"

// Ein Sensor-Objekt erstellen
BMP581 pressureSensor;

// I2C Adresse aussuchen. Das ELV-AM-AP hat 0x46 als Standard-Adresse
//uint8_t i2cAddress = BMP581_I2C_ADDRESS_DEFAULT; // 0x47
uint8_t i2cAddress = BMP581_I2C_ADDRESS_SECONDARY; // 0x46

void setup()
{
    // Serielle Schnittstelle initialisieren
    Serial.begin(115200);
    Serial.println("BMP581 Example1 begin!");

    // I2C Bibliothek initialisieren
    Wire.begin();

    // Sensor testen,
    while(pressureSensor.beginI2C(i2cAddress) != BMP5_OK)
    {
        // Falls nicht verbunden, Ausgabe
        Serial.println("Error: BMP581 not connected, check wiring and I2C
address!");

        // Etwas abwarten und dann noch einmal probieren
        delay(1000);
    }

    Serial.println("BMP581 connected!");
}

void loop()
{
    // Sensordaten auslesen
    bmp5_sensor_data data = {0,0};
    int8_t err = pressureSensor.getSensorData(&data);

    // Fehler auslesen, ob Auslesen erfolgreich war
    if(err == BMP5_OK)
    {
        // Auslesen erfolgreich, Ausgabe der Messwerte
        Serial.print("Temperature (C): ");
        Serial.print(data.temperature);
        Serial.print("\t\t");
        Serial.print("Pressure (Pa): ");
        Serial.println(data.pressure);
    }
    else
    {
        // Auslesen nicht erfolgreich, Fehler ausgeben
        Serial.print("Error getting data from sensor! Error code: ");
        Serial.println(err);
    }

    // Nur jede Sekunde ausgeben.
    delay(1000);

}
```

Codeblock 6

```
#include <STM32LoRaWAN.h>
#include <STM32LowPower.h>
#include <Wire.h>
#include "SparkFun_BMP581_Arduino_Library.h"

STM32LoRaWAN modem;

// Create a new sensor object
BMP581 pressureSensor;

// I2C address selection
uint8_t i2cAddress = BMP581_I2C_ADDRESS_SECONDARY; // 0x46

static const unsigned long TX_INTERVAL = 60000; /* ms */
unsigned long last_tx = 0;
unsigned int counter = 0;

/* Get the rtc object */
STM32RTC& rtc = STM32RTC::getInstance();

void setup() {
  LowPower.begin();
  Serial.begin(115200);
  Serial.println("Start");
  modem.begin(EU868);

  // Configure join method by (un)commenting the right method
  // call, and fill in credentials in that method call.
  bool connected = modem.joinOTAA(/* AppEui */ "00000000000000", /* AppKey */
"00000000000000000000000000000000", /* DevEui */ "00000000000000");
  //bool connected = modem.joinABP(/* DevAddr */ "00000000", /* NwkSKey */
"00000000000000000000000000000000", /* AppSKey */
"00000000000000000000000000000000");

  if (connected) {
    Serial.println("Joined");
  } else {
    Serial.println("Join failed");
    while (true) /* infinite loop */
      ;
  }

  Wire.begin();

  /* set the calendar */
  rtc.begin(true);
  rtc.setTime(00, 00, 00);
  rtc.setDate(00, 00, 00);

  while(pressureSensor.beginI2C(i2cAddress) != BMP5_OK)

    {
        // Not connected, inform user
        Serial.println("Error: BMP581 not connected, check wiring and I2C
address!");

        // Wait a bit to see if connection is established
        delay(1000);
    }

    Serial.println("BMP581 connected!");
}

void send_packet() {
  counter++;
  bmp5_sensor_data data = {0,0};
  int8_t err = pressureSensor.getSensorData(&data);

  Serial.println(data.temperature);
  Serial.println(data.pressure);
  char payload[50] = { 0 }; /* packet to be sent */
  /* prepare the Tx packet : get date and format string */
  String temp = String(data.temperature);
  String pres = String(data.pressure);
  sprintf(payload, "#%d: %02d/%02d/%04d - %02d:%02d:%02d: %s C %s Pa",
          counter,
          rtc.getMonth(), rtc.getDay(), 2000 + rtc.getYear(),
          rtc.getHours(), rtc.getMinutes(), rtc.getSeconds(), temp.c_str(),
pres.c_str());
  modem.setPort(10);
  modem.beginPacket();
  modem.write(payload, strlen(payload));
  if (modem.endPacket() == (int)strlen(payload)) {
    Serial.println("Sent packet");
  } else {
    Serial.println("Failed to send packet");
  }

  if (modem.available()) {
    Serial.print("Received packet on port ");
    Serial.print(modem.getDownlinkPort());
    Serial.print(":");
    while (modem.available()) {
      uint8_t b = modem.read();
      Serial.print(" ");
      Serial.print(b >> 4, HEX);
      Serial.print(b & 0xF, HEX);
    }
    Serial.println();
  }
  Serial.flush();
}

void loop() {
    send_packet();
    LowPower.deepSleep(TX_INTERVAL);
    Serial.begin(115200);
    Wire.begin();

}
```