AUFGABENBLATT: SMARTE CHRISTBAUMKUGEL

T-Systems on site services GmbH

Aufgabenblatt: smarte Christbaumkugel

Version:	1.00	Gültig ab:	01.11.2019

Mit Veröffentlichung dieses Dokumentes verlieren alle bisherigen Versionen ihre Gültigkeit!

T · · Systems ·

IMPRESSUM

Herausgeber	T-Systems on site Services GmbH Fasanenweg 5 70771 Leinfelden-Echterdingen

Version	**Letztes Review**
1.00	01.11.2019

Kurzbeschreibung
Anweisung und Hilfestellung zur Umsetzung der IoT-Christbaumkugel.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 CHRISTBAUMKUGEL

1.1 Aufgabe

Ziel dieses Workshops ist das Entwickeln einer smarten Christbaumkugel, welche per Bluetooth Low Energy steuerbar ist. Hierzu wird ein ESP32-Mikrocontroller mit zwei angeschlossenen LED-Stripes genutzt, wodurch Unmengen an Farbspielen möglich werden, welche für die weihnachtliche Stimmung sorgen! Das Lichtspiel der LEDs erstellt ihr in eurem Programm ganz nach euren Wünschen selbst. Um zwischen den verschiedenen Lichtspielen zu wechseln könnt ihr den ESP32 per Bluetooth mit eurem Handy verbinden und direkt eure verschiedenen Programme ansteuern. Abschließend wird die gesamte Hardware noch in eine Plastikkugel verpackt, und ihr habt die Möglichkeit, sie individuell nach euren Wünschen zu gestalten.

1.2 Schaltplan

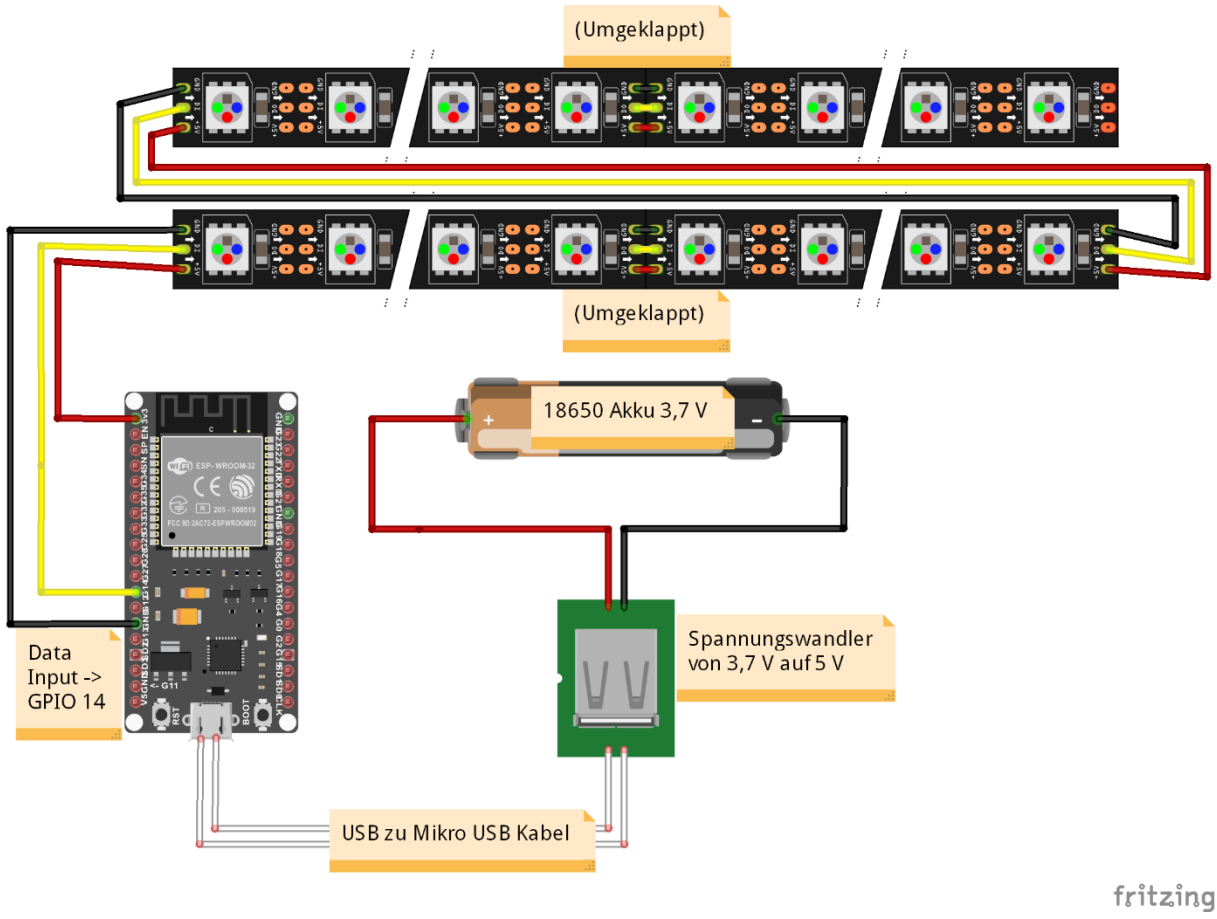

Abbildung 1: Schematischer Schaltplan Christbaumkugel

1.2.1 Hardware Komponenten

ESP 32

Der verwendete Mikro Der ESP32 ist eine kostengünstige und mit geringem Leistungsbedarf ausgeführte 32-Bit-Mikrocontrollerfamilie der Firma espressif.

LED Stripe

WS2812b LED Stripe. Bei den WS2812 LEDs handelt es sich um adressierbare RGB-LEDs. Sie verfügen über einen integrierten Chip und belegen daher nur einen einzigen digitalen Output

Spannungswandler

Wird benötigt, um die 3,7 Volt des Akkus auf konstante 5 Volt zu bringen, so kommt es nicht zu Verbindungsabbrüchen des BLE Chips. Der Spannungswandler verfügt über einen USB Output, wodurch eine einfache Stromversorgung des ESP32 ermöglicht wird.

18650 Akku

Lithium Ionen Akku zur Stromversorgung der Christbaumkugel. Der Akku liefert 3,7 Volt und bietet eine ausreichende Kapazität von 3.450mAh.

WICHTIG: Wenn die Kugel nicht verwendet wird, sollte der Akku entfernt werden!

1.3 Benötigte Bibliotheken

Um den Programmcode für unsere Christbaumkugel zu schreiben werden einige externe Bibliotheken benötigt, welche mittels „include“ eingebunden werden müssen.

Folgende Bibliotheken sind bereits durch Arduino selbst oder durch das „esp32“-Paket mitgeliefert:

- **ESP_WiFi:** Stellt allgemeine Funktionen für WiFi-Verbindungen zu Verfügung. Wir benötigen sie aber nur um WiFi abzuschalten, damit der Stromverbrauch reduziert wird.
- **BLEDevice:** Klasse für ein BLE Device. Die Grundlage für die BLE Kommunikation.
- **BLEServer:** Klasse für einen BLE Server. Hierauf kann sich später das Smartphone verbinden.
- **BLEUtils:** Allgemeine Funktionen für BLE.

Um die LED-Stripes anzusteuern wird zusätzlich die „NeoPixel“-Bibliothek von Adafruit benötigt. Eigentlich ist diese Bibliothek dafür gedacht einen LED-Ring von Adafruit anzusteuern, aber sie funktioniert auch wunderbar mit unseren LED-Stripes.

Um sie zu installieren muss diese zunächst aus den Git-Repository von Adafruit heruntergeladen werden: https://github.com/adafruit/Adafruit_NeoPixel. Das ZIP-Paket kann nun an einem beliebigen Ort auf dem Computer ablegen werden. Als nächstes muss das Paket in die Arduino IDE eingebunden werden. Dies funktioniert wie folgt: „Sketch -> Bibliothek einbinden -> ZIP-Bibliothek hinzufügen…“ anschließend das heruntergeladene Paket auswählen.

Das Paket liefert zusätzlich zur Bibliothek einige Beispiele, die in der IDE unter „Datei -> Beispiele -> Adafruit NeoPixel“ zu finden sind.

1.4 nRFConnect App

Um die Bluetooth-Verbindung von Smartphone aus zu testen kann eine beliebige Bluetooth App verwendet werden. Für diesen Workshop wird die App nRFConnect genutzt. Sie ist für Android sowie iOS kostenlos verfügbar:

- https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en_US
- https://apps.apple.com/us/app/nrf-connect/id1054362403

1.5 Entwicklung mit Arduino

Um das Board über den Computer programmieren zu können, wird es per Mikro-USB angeschlossen. Das Board wird über einen virtuellen COM Port angesprochen. Hierzu benötigt der Computer noch die passenden Treiber. Diese sind hier zu finden:

https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

Anschließend kann man sich die Arduino IDE unter https://www.arduino.cc/ für alle gängigen Betriebssysteme downloaden und installieren. Zunächst ist es wichtig, alle benötigten Boardkonfigurationen und Bibliotheken einzubinden, um den ESP32 über die IDE nutzen zu können. Dazu geht man unter „Datei -> Voreinstellungen“ und fügt eine „zusätzliche Boardverwalter-URL“ ein: *https://dl.espressif.com/dl/package_esp32_index.json*

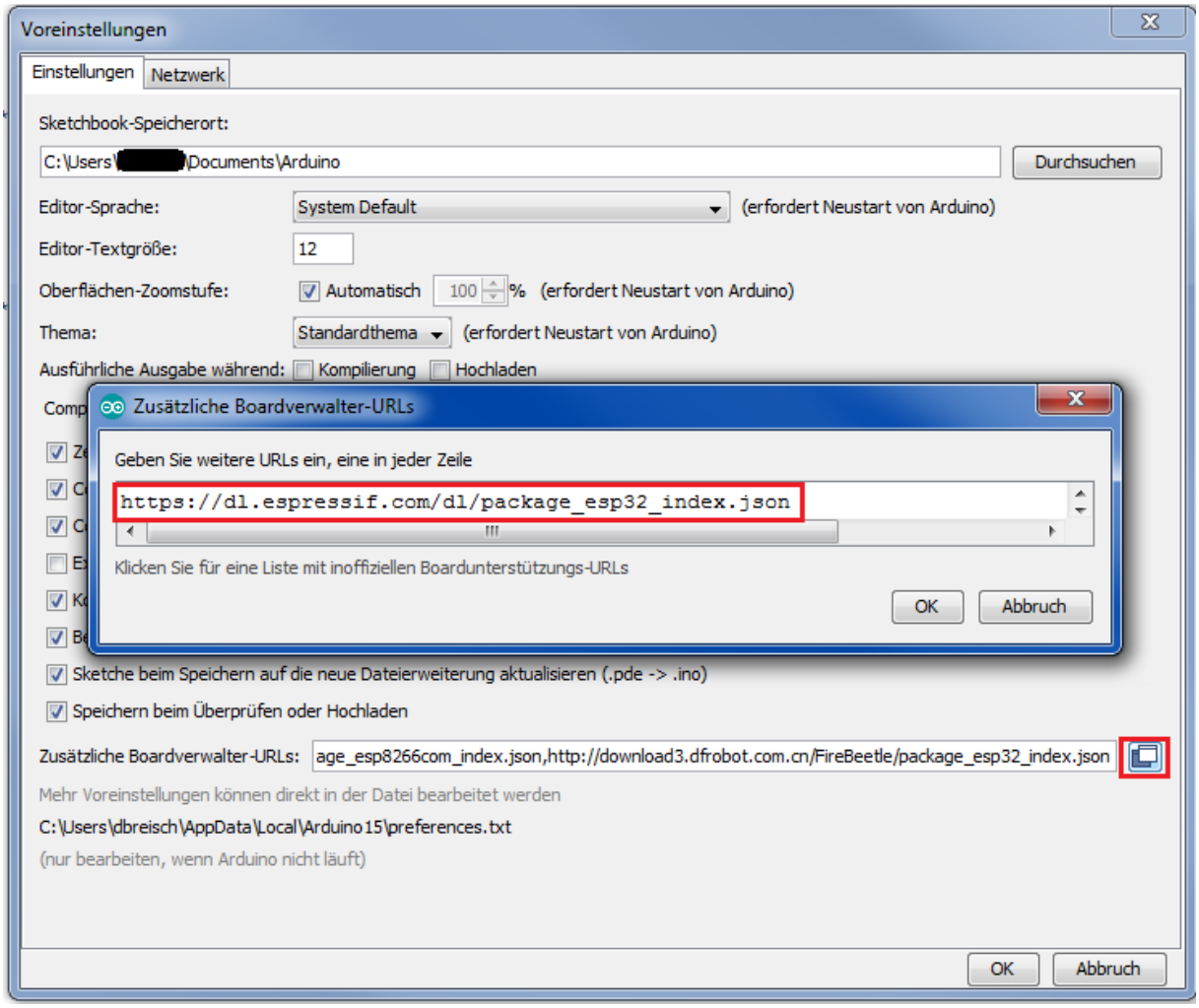

Abbildung 2: Zusätzliche Boardverwalter hinzufügen

Nun kann unter „Werkzeuge -> Board -> Boardverwalter“ das „esp32“ Paket installiert werden.

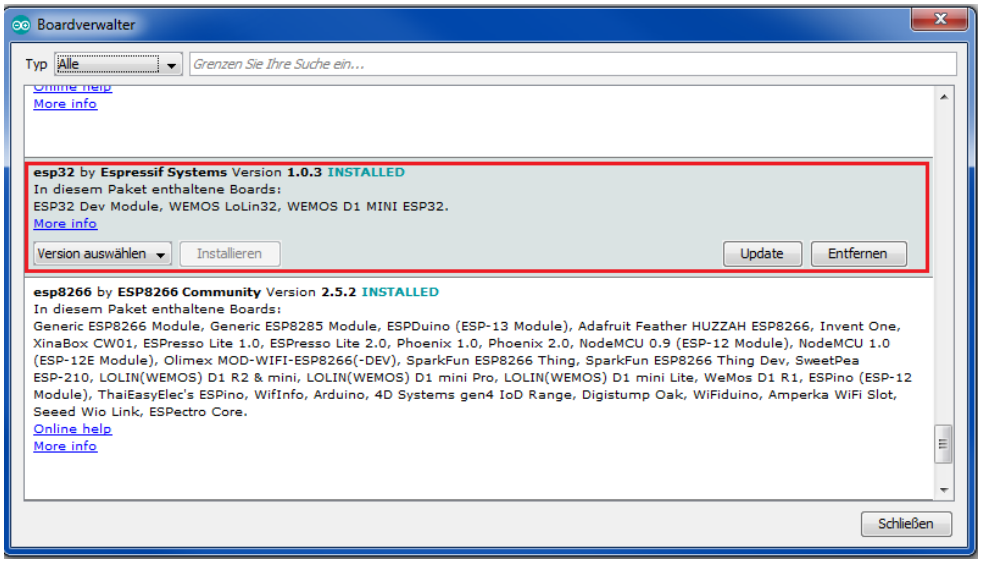

Abbildung 3: esp32 Paket installieren

Damit nun die IDE weiß für welches Board tatsächlich entwickelt wird, gibt man abschließend unter „Werkzeuge“ die korrekte Konfiguration wie in Abbildung 4 dargestellt an. Wichtig hierbei ist zudem das Auswählen des richtigen virtuelle COM Ports, an dem das Board angeschlossen ist.

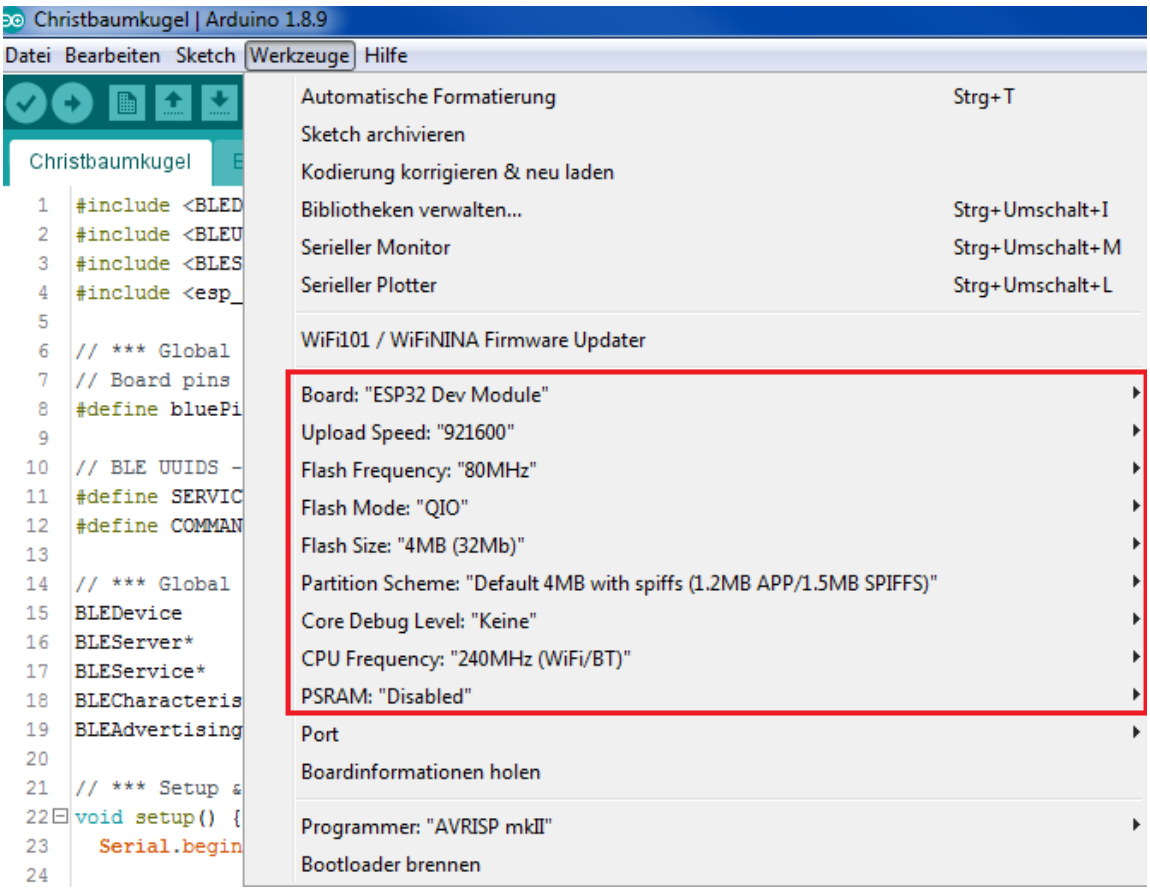

Abbildung 4: Boardkonfiguration

Will man den Sketch auf das Board flashen, klickt man auf den unten markierten Pfeil „Hochladen“. Daraufhin wird der Sketch kompiliert und das Flashen beginnt. Will man lediglich kompilieren, klickt man auf den Haken „Überprüfen“.

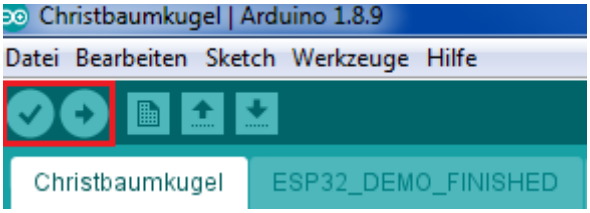

Abbildung 5: Buttons "Überprüfen" und "Hochladen"

Während des Flashvorgangs wird am unteren Bildschirm der aktuelle Vorgang in der Konsole ausgegeben. Sobald dort „Connecting…“ steht, muss der „Boot“ Knopf des Boards gedrückt werden, bis die Übertragung startet. Danach kann der Knopf losgelassen werden.

Abbildung 6: Konsolenausgabe beim Flashen

Wenn die IDE die Übertragung abgeschlossen hat, muss abschließend noch der „Reset“ Knopf gedrückt werden, um das neue Programm zu starten.

Eine praktische Debugmöglichkeit bietet der „Serielle Monitor“, der in der rechten oberen Ecke zu finden ist. Hier werden alle seriellen Ausgaben des Controllers angezeigt, die der Entwickler erstellt hat.

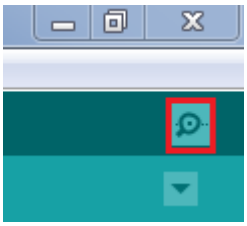

Abbildung 7: Button "Serieller Monitor"

Für Linux Nutzer:
Unter Linux muss zunächst die Versionskontrollsoftware Git und das Python-Modul Serial mittels

```
sudo apt install git python-serial
```

installiert wesrden.

Damit nun die IDE weiß für welches Board tatsächlich entwickelt wird, gibt man abschließend unter „Werkzeuge“ die korrekte Konfiguration wie in Abbildung 4 dargestellt an. Wichtig hierbei ist zudem das Auswählen des richtigen virtuelle COM Ports, an dem das Board angeschlossen ist.

1.5.1 Beispiel: Hello World

Zum Testen der Konfiguration kann ein kleiner „Hello-World“ Sketch implementiert werden. Hierzu muss über „Datei -> Neu“ ein neuer Sketch angelegt werden. Mithilfe des in Abbildung 8

dargestelltem Sketch kann eine Serielle Ausgabe geschrieben werden, welche im Seriellen Monitor verfolgt werden kann. Dabei ist darauf zu achten, dass die Baudrate im Seriellen Monitor mit der im Sketch hinterlegten Baudrate (115200) übereinstimmt.

```
void setup() {
 Serial.begin(115200);
 Serial.println("Hello ESP32 World!");
}

void loop() {
Serial.println("Hello");
delay(500);
}
```

Abbildung 8 Hello World Sketch

2 EINZELNE ARBEITSSCHRITTE

1. **Einbinden aller nötigen Bibliotheken**
   Um die externen Bibliotheken nutzen zu können, müssen diese zu Beginn eingebunden werden.

```
#include <Adafruit_NeoPixel.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <esp_wifi.h>
```

2. **Präprozessor Konstanten und Variablen Definieren**

   Damit verschiedene Werte, wie ein Port Pin oder die Anzahl der LEDs neu eingetippt werden müssen, können jetzt Präprozessor Konstanten und Variablen gesetzt werden.

```
#define LED_PIN 14
#define LED_NUMBER 8
#define BRIGHTNESS 100

BLEDevice DeviceM;
BLEServer* pServer;
BLEService* pService;
BLECharacteristic* pCharCommand;
BLEAdvertising* pAdvertising;

Adafruit_NeoPixel stripe(LED_NUMBER, LED_PIN, NEO_GRB +
NEO_KHZ800);
```

   Gegebenenfalls müssen später noch weitere Variablen hinzugefügt werden, dies ist das Grundsetup, welches für die Bearbeitung der Aufgabe benötigt wird.

3. **Serielle Ausgabe initialisieren, um über den Seriellen Monitor debuggen zu können.**
   Vor der Verwendung der seriellen Schnittstelle muss diese im SETUP-Teil des Arduino-Sketches initialisiert werden. Die Baudrate gibt dabei die Übertragungsgeschwindigkeit an und muss bei Sender und Empfänger übereinstimmen.

4. **Bluetooth initialisieren**
   Um eine Verbindung mit dem Central einzugehen, muss zunächst das Bluetooth initialisiert werden. Dazu sind mehrere Schritte nötigt.

   a. Bluetooth Gerät erstellen

   Zunächst muss das BLE Gerät initialisiert werden. Dabei kann der Name des Gerätes bestimmt werden.

```
Device.init("Name of BLE Device");
```

   b. BLE Server erstellen

   Nachdem das BLE Gerät erstellt wurde, muss dieses als BLE Server konfiguriert werden. `pServer` ist dabei vom Typ `BLEServer`:

```
pServer = Device.createServer();
```

   c. Profil definieren

   Ein mögliches Profil, bestehend aus einem Service, könnte wie in folgender Abbildung dargestellt aufgebaut werden. Die „Controll Characteristic“ wird verwendet, um die LEDs

ein- bzw. auszuschalten. Die „LED-Mode Characteristic“ kapselt den Wert des aktuellen LED-Modes. Durch das Verändern des Charakteristik Wertes, kann somit das Lichtspiel der Kugel gewechselt werden.

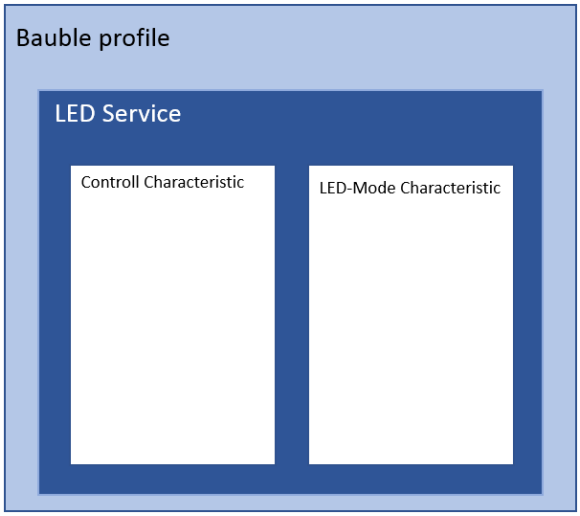

Abbildung 9 Bauble profile

d. UUIDs erstellen

Der Service sowie die Charakteristiken benötigen eine UUID, also eine einzigartige ID welche sie repräsentiert. Wie unter Aufgabenpunkt 2 werden dafür Präprozessor Konstanten angelegt. Diese werden direkt unter die oben angelegten Konstanten geschrieben. Unter https://www.uuidgenerator.net/ können UUIDs generiert werden. Für jeden Service und für jede Charakteristik muss je eine UUID generiert und in einer Präprozessor Konstante als String gespeichert werden.

e. In den Advertising Zustand wechseln

Ist das Gerät initialisiert und das Profil definiert, kann der Service gestartet und in den Advertising Zustand gewechselt werden, sodass andere BLE Geräte das Gerät durch das scannen der Umgebung finden können. Die nötigen Befehle können dem Cheat sheet entnommen werden.

Anschließend versendet das Gerät Advertising Nachrichten mit der definierten Nachricht auf den dafür vorgesehen Advertising Kanälen. Durch das scannen der Umgebung mithilfe der NRFConnect App, sollte das initialisierte Gerät nun sichtbar werden. Dazu muss der Sketch zunächst kompiliert und auf dem Mikrocontroller geflasht werden.

f. Callback Methode definieren

Ein Kernkonzept von BLE ist der Begriff der Charakteristik. Diese kann als zustandsbehafteten Datensatz betrachtet werden, der eine Identität und einen Wert hält. Ein BLE Client kann den Wert der Charakteristik lesen oder einen neuen setzen (falls zulässig). Der Wert der Charakteristik liefert demnach die Informationen. Wenn die Charakteristik z.B. die Herzfrequenz repräsentiert, gemessen von einem Sensor, dann kann ein entfernter Client die aktuelle Herzfrequenz abrufen, indem er den aktuellen Wert der Charakteristik liest. In diesem Fall wird ein Remote-Client den Wert nicht neu schreiben. Der Wert wird ausschließlich intern vom BLE-Server geändert, entweder jedes Mal, wenn eine Leseanforderung von einem Client gestellt wird oder wenn ein neuer Wert vom Sensor gemessen wird.

Um nun auf einen solchen Zugriff der Charakteristik reagieren zu können muss jeweils eine entsprechende Callback Methode registriert werden. Die Klasse namens BLECharacteristicCallbacks bietet hierfür zwei Methoden, die überschrieben werden können:

`onRead(BLECharacteristic* pCharacteristic)` – Wird aufgerufen, wenn eine Leseanforderung von einem Client eingeht. Ein neuer Wert für das Merkmal kann vor der Rückkehr aus der Funktion gesetzt werden und wird als der vom Client empfangene Wert verwendet.

`onWrite(BLECharacteristic* pCharacteristic)` – Wird aufgerufen, wenn eine Schreibanfrage eines Clients eingeht. Der neue Wert wurde bereits im Merkmal gesetzt und Ihre eigene Codelogik kann den Wert lesen und eine Aktion durchführen.

Info: Eine Callback-Funktion ist eine Funktion, die einer anderen Funktion als Parameter übergeben und von dieser unter gewissen Bedingungen aufgerufen wird.

Ähnlich wie bei den Callbacks einer Charakteristik gibt es auch Server-Callbacks. Diese informieren den Code über Verbindungs- und Trennungsereignisse beim Client. Diese sind der BLEServerCallbacks-Klasse untergeordnet, für die es virtuelle Methoden gibt:

`onConnect(BLEServer* pServer)` - Wird aufgerufen, wenn eine Verbindung hergestellt wird.

`onDisconnect(BLEServer* pServer)` - Wird aufgerufen, wenn eine Trennung stattfindet.

5. Verbindungstest mit dem Smartphone durchführen

Um die BLE Konfiguration zu testen, kann nun eine Verbindung zum Erstellen BLE Gerät hergestellt werden. Der definierte Name sollte Dazu nach dem Scannen in der Liste der gefundenen Geräte sichtbar werden. Nach dem erfolgreichen Verbindungsaufbau ist das definierte Profil nun auf der Oberfläche sichtbar.

Durch einen Klick auf den definierten Service, können die gekapselten Charakteristiken ausgeklappt werden. Um sicherzustellen, dass alles funktioniert kann der definierte default-Wert der Charakteristik ausgelesen werden.

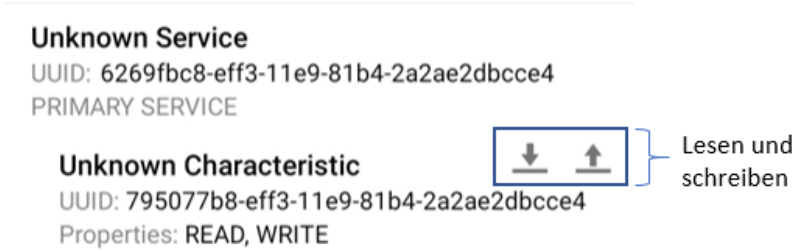

Abbildung 10 Lesen und Schreiben einer Charakteristik mithilfe der App nRF Connect

6. LED-Stripe initialisieren und verschiedene Lichtspiele programmieren

Zunächst muss mit folgender Zeile eine Neo-Pixel Objekt initialisiert werden. Dabei müssen mehrere Parameter definiert werden:

- Die Anzahl der verwendeten LEDs auf dem Stripe. Diese können im folgendem über einen Index angesprochen werden.
- Die Pin-Nummer an dem der LED-Stripe am Mikrocontroller angeschlossen ist.
- Ein Wert, der den Typ der angeschlossenen NeoPixel angibt

```
Adafruit_NeoPixel stripe(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
```

Anschließend kann in der setup() Funktion begin() aufgerufen werden, um den Datenpin für die NeoPixel-Ausgabe vorzubereiten.

```
stripe.begin();
```

Optional kann hier auch die Helligkeit der LEDs definiert werden. Dies kann allerdings auch für jedes Lichtmuster separat definiert werden. Die Helligkeit wird wie folgt konfiguriert:

```
stripe.setBrightness(BRIGHTNESS);
```

Info: Durch die Reduzierung der Helligkeit, kann der Lebensdauer des Akkus positiv beeinflusst werden 😊

Durch folgenden Code kann nun die Farbe einer LED an einem spezifischen index definiert werden. Durch show() wird die definierte Pixelfarbe an den Stripe gesendet, sodass sich diese aktualisiert.

```
stripe.setPixelColor(index, color);
stripe.show();
```

Durch das separate Ansprechen jeder einzelnen LED können nun beliebige Lichtspiele entworfen werden.

Tipp: Lasst euch durch die mitgelieferten Beispiele der Bibliothek inspirieren und seit kreativ.

7. Lichtspiele über Bluetooth ansteuern

Damit die implementierten Lichtspiele über BLE gesteuert werden können, muss mithilfe der Callback-Funktion der gesendete Wert ausgelesen und entsprechend verarbeitet werden.

A CHEAT SHEET

A.1 Allgemein

Serial.begin(9600);	Baudrate für die serielle Übertragung definieren.
Serial.print(“STRING“); Serial.println(“STRING“);	Gibt STRING auf dem seriellen Monitor aus. Ohne und mit Zeilenumbruch. Gut für Debugging.
#include <BIBLIOTHEK.h>	Einbinden einer Bibliothek. Immer ganz am Anfang des Sketches.
#define NAME 12	Definiert die Variable NAME für die Zahl „12“. Nützlich zum globalen definieren von z.B. Pinnummern.
delay(int MILLISEKUNDEN);	Pausiert das Sketch für MILLISEKUNDEN.

A.2 LED-Stripe

Adafruit_NeoPixel stripe(LED_ANZAHL, LED_PIN, NEO_GRB + NEO_KHZ800);	Erstellt das Objekt “stripe” vom Typ “Adafruit_NeoPixel”. Parameter: - LED_ANZAHL: Anzahl der LEDs auf dem Stripe. - LED_PIN: Date input Pin am Board. - NEO_GRB + NEO_KHZ800: Konstanter Wert.
stripe.begin();	Startet den LED-Stripe.
stripe.show();	Änderungen am LED-Stripe übernehmen. Z.B. bei neuer Farbe.
stripe.setBrightness(int HELLIGKEIT);	Setzt die Helligkeit der LEDs. Wert von 0 – 255.
stripe.Color(RED, GREEN, BLUE);	Erstellt Farbcode anhand von RGB-Code. Gibt den Farbcode als uint32_t zurück.
stripe.setPixelColor(int POSITION_LED, uint32_t FARBE);	Setzt LED an POSITION_LED auf Farbe FARBE.
stripe.fill(uint32_t FARBE, VonLED, BisLED);	Setzt mehrere LEDs auf Farbe FARBE
stripe.clear();	Schaltet alle LEDs aus. Funktioniert nur in Verbindung mit stripe.show().

A.3 Bluetooth

Device.init("BLE_Kugel");
pServer = Device.createServer();
pService = pServer->
createService(SERVICE_UUID);
	Erstellt ein BLE Device namens „BLE_Kugel“, fügt einen Server sowie einen Service mit einer UUID hinzu.
	Typ: BLEDevice

//Deklaration BLECharacteristic* pCharCommand; BLEServer* pServer; pCharCommand = pService-> createCharacteristic(COMMAND_CHAR_UUID, BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_READ);	Erstellt eine Charakteristik für den Service pService. Parameter: - COMMAND_CHAR_UUID: Die UUID der Charakteristik. - PROPERTY_WRITE: Die Charakteristik hat Schreibzugriff. PROPERTY_READ: Die Charakteristik hat Lesezugriff.
pCharCommand->setValue("default");	Setzt den aktuellen Wert der Charakteristik auf „default“. Dieser Wert kann dann vom Smartphone ausgelesen oder geändert werden.

//Deklaration
BLEService* pService;
 	
pService->start();
	Startet den Service.

//Deklaration
BLEAdvertising* pAdvertising;
 	
pAdvertising = Device.getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pAdvertising->setScanResponse(true);
pAdvertising->setMinPreferred(0x06);
pAdvertising->setMinPreferred(0x12);
Device.startAdvertising();
	Konfiguriert und startet das BLE Advertising. Notwendig, um von anderen Geräten gesehen zu werden.

pServer->setCallbacks(new ServerCallbacks()); pCharCommand->setCallbacks(new CharacteristicCallbacks());	Callbacks für den Server und die Charakteristik registrieren.

class ServerCallbacks : public BLEServerCallbacks
{
void onConnect(BLEServer* pServerCallback) {
Serial.println("Client connected");
}
};
	Beispiel Server-Callback, der bei Verbindung eines Clients eine Meldung auf dem seriellen Monitor ausgibt.

class CharacteristicCallbacks : public
BLECharacteristicCallbacks {
void onWrite(BLECharacteristic* pCharacteristic){
std::string value = pCharacteristic->getValue();
Serial.println(value);
}
};
	Beispielhafter Charakteristik Callback der die von einem BLE-Gerät übermittelte Daten auf dem seriellen Monitor ausgibt.

Color uint_32t

Characteristicvalue std::string

CHARACTERISTIC -> getValue();

std::string -> string (std::string.c_str())

String -> int (string.toInt())