ARBEITSBLATT: IOT AM WEIHNACHTSBAUM

IOT XMAS WORKSHOP @ CMD+O

T-Systems on site Services GmbH

Version: 1.0

IOT XMAS WORKSHOP @ CMD+O

T · · Systems ·

IMPRESSUM

Tabelle 1 Impressum

Herausgeber	T-Systems onsite Services GmbH Fasanenweg 5 70771 Leinfelden-Echterdingen

Version	Letztes Review	Status
1.00	20.11.2021	

Kurzbeschreibung
IoT am Weihnachtsbaum: Arbeitsblatt zur Umsetzung

Inhaltsverzeichnis

Abbildungsverzeichnis

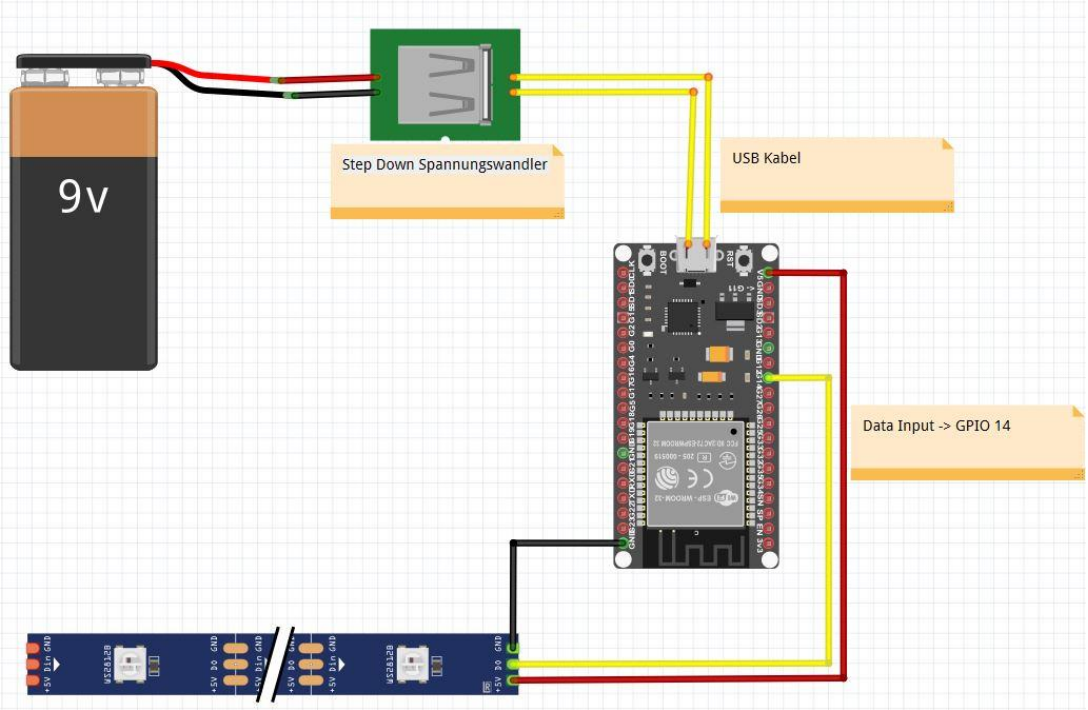
Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

1 IOT AM WEIHNACHTSBAUM

1.1 Was ist zu tun?

Ziel ist, das Entwickeln deiner eigenen IoT Christbaumkugel, welche sich per Bluetooth Low Energy steuern lässt. Das Herzstück ist ein ESP32 Dev Board, welches du über die Arduino IDE programmieren kannst. Für die Kommunikation zwischen App und T-Racer wird das Protokoll Bluetooth Low Energy eingesetzt.

1.2 Der Schaltplan



1.3 Hardware Komponenten

ESP 32

Der ESP32 ist eine kostengünstige und mit geringem Leistungsbedarf ausgeführte 32-Bit-Mikrocontrollerfamilie der Firma espressif. Er kommt standardmäßig mit Wifi und BLE an Bord.

USB Step down module

Wird benötigt, um die 9V unserer Batterie auf konstante 5V zu bringen, um eine einfache Stromversorgung zu gewährleisten. Das Step down modul wandelt die Eingangsspannung (6V - 24V) in einen festen 5V-USB-Ausgang mit bis zu 3A Dauerausgangsstrom um.

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. Umgangssprachlich formuliert - Du kannst jeder einzelnen LED sagen, was sie zu tun hat.

2 PROGRAM YOUR CODE

Im jeweiligen Base-sketch (Expert/Beginner) ist bereits ein Grundgerüst implementiert, welches vervollständigt werden muss. Die fehlenden code-snippets sind jeweils mit einem „TODO“ Kommentar, sowie einer kleinen Beschreibung als Kommentar vorhanden.

INFO: Nach jedem Schritt kann der Code mithilfe der NrfConnect App überprüft werden.

2.1 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.

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.

2.2 Bluetooth LE initialisieren

Um eine Verbindung mit dem Central einzugehen, muss zunächst das Bluetooth initialisiert werden. Dazu sind mehrere Schritte nötigt, welche in der Methode `initBLE()` durchgeführt werden. Die Initialisierung wird zu Beginn einmalig in der Setup() Funktion aufgerufen.

Zunächst müssen die BLE variablen definiert werden, diese sind bereits im Base-sketch vorhanden:

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

a. Bluetooth Gerät erstellen

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

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

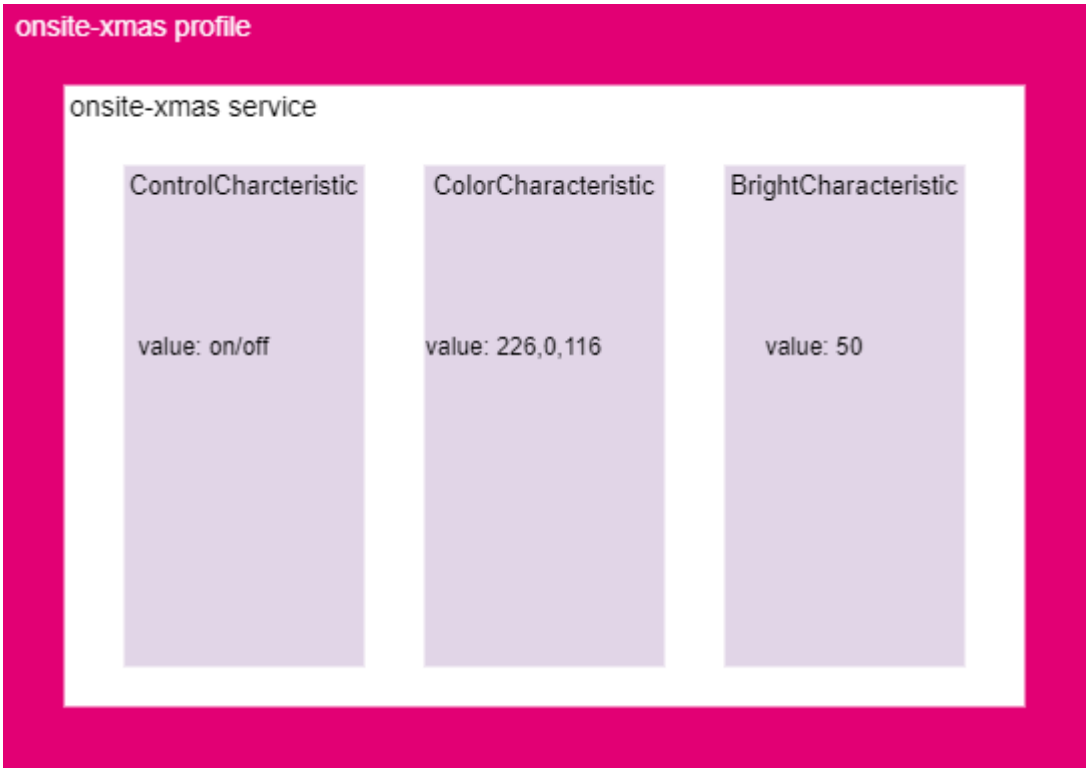
b. BLE Server erstellen

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 „Color Characteristic“ wird genutzt, um die Farbe in Form von RGB zu schreiben. Mithilfe der BrightCharacteristic kann die Helligkeit der Christbaumkugel bestimmt werden.



Dieses muss im TODO BLE PROFIL umgesetzt werden.

d. UUIDs erstellen

Der Service sowie die Charakteristiken benötigen eine UUID, also eine einzigartige ID welche sie repräsentiert. Diese wurden bereits als Konstanten angelegt. Unter https://www.uuidgenerator.net/ können UUIDs generiert werden. Für jeden Service und für jede Charakteristik wurde je eine UUID generiert Diese müssen nun den Services/Characteristiken zuegordnet werden.

e. 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.

Unter dem TODO: Callback Methoden definieren, müssen nun die Callbacks registriert werden. Der ServiceCallback wurde bereits im Base-Sketch registriert, sowie die onConnect/onDisconnect Methoden überschrieben.

Ähnlich funktioniert dies auch mit den Characteristic Callbacks. (Im Beginner Sketch wurden bereits alle Callback Methoden implementiert, diese müssen jedoch noch in der initBle() registriert werden)

→TODO LedColorCallback Methode (Kapitel 2.4)

→TODO BrightnessCallback Methode (Kapitel 2.5)

f. Advertising

Anschließend wird in der Methode initBLE() in den Advertising zustand gewechselt, sodass andere die Christbaumkugel finden können.

2.3 Turn on and off

Um Alle LEDs auszuschalten muss in der Methode `turn_off_leds()` alle rgb-Werte auf 0 gesetzt werden. Dazu kann die Methode setColor(String rgb) genutzt werden. (TODO LEDs ausschalten). Dies Methode zum ausschalten muss zusätzlich noch in der überschrieben `onWrite()` Methode der Callback Funktion der ControlCharacteristic aufgerufen werden, sofern `check_on_off(String powerMode)` *false* zurück gibt.

TODO ControlCharacteristicCallback Methode:

Gibt die Methode `check_on_off(String powerMode)` jedoch *true* zurück, müssen die LEDs eingeschalten werden. Dazu kann die Methode `show(uint32_t color)` aufgerufen werden.

Der Code sollte dann in etwa so aussehen:

```
        if (check_on_off(powerMode.c_str())) {
          Serial.print("Turn on");
          show(Stripe.Color(226, 0, 116));
        } else {
          Serial.print("Turn off");
          turn_off_leds();
        }
```

Die Methode `c_str()` konvertiert den Inhalt eines std:strings in eine null-terminierte Zeichenkette (String) im C-Stil.

Sobald Farbwerte gesetzt werden, wird die Christbaumkugel „eingeschalten“.

INFO: Die Methode show() wird erst im Kapitel 2.4 vollständig implementiert.

INFO: Die onWrite() Callback Methoden können wie folgt überschrieben werden:

```
class ControlCharacteristicCallback: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic* pCharacteristic) {
    if (pControlCharacteristic->getValue().length() > 0) {
      std::string powerMode = pControlCharacteristic->getValue();
    }
  }
};
```

Wird die Verbindung Getrennt oder ein Gerät erfolgreich Verbunden, kann die Christbaumkugel aus bzw. eingeschalten werden. Dazu muss die ServreCallback onDisconnect und onConnect überschrieben werden:

TODO Einschalten

```
show(Stripe.Color(226, 0, 116));
```

TODO Ausschalten

```
turn_off_leds();
```

2.4 Farbe ändern

Um Die Farbe zu ändern und zu setzen, kann die `setColor(String rgb)` Methode erweitert werden.

TODO Parsen: Der String, welcher in die Characteristic geschrieben wird, muss hier in die RGB Werte gesplittet werden. Anschließend müssen die Werte in den Datentyp *int* konvertiert werden.

Hierzu kann die Methode `substring(int, int)` der Klasse String genutzt werden.
Beispiel: `r   = rgb.substring(0, firstSplit).toInt();`

Die Werte r,g und b wurden bereits als globale variablen definiert und können hier überschrieben werden.

Anschließend müssen die Werte noch auf den Neopixel (Stripe) übertragen werden.

TODO RGB-Werte übertragen: hier reicht es die Methode `show(uint32_t color)` aufzurufen.

→ `show(Stripe.Color(r, g, b));`

TODO Farben setzen: Um alle Pixel auf dem LED-Stripe zu setzen, muss über jeden einzelnen iteriert werden, um die gewünschte Farbe zuzuweisen. Für die Zuweisung des RGB Farbcodes muss einmal die Farbe auf dem gewünschten Pixel gesetzt werden und anschließend mit der Methode `show()` auf die „Hardware“ übertragen werden.

```
Stripe.setPixelColor(i, color);
Stripe.show();
```

Um Die Farbe der Christbaumkugel nun mit dem Handy über BLE zu steuern, müssen wir die Methode `setColor(String rgb)` in der überschiebenen onWrite() Methode der ColorCharacteristic aufrufen. Der Code dazu sieht wie folgt aus (TODO LedColorCallback Methode):

```
    if (isOn) {
        // Get value via Bluetooth
        std::string  ledColor = pColorCharacteristic->getValue();
        Serial.print("New LED Color: ");
        Serial.println(ledColor.c_str());
        setColor(ledColor.c_str());
      }
```

2.5 Helligkeit anpassen

Um die Helligkeit via BLE anpassen zu können muss die Methode des Basis-sketches `setBright(std::string bright)` überschrieben werden (TODO Helligkeit). Dazu muss lediglich die Methode `setBrightness()` der Neopixel Bibliothek aufgerufen werden. Um den übergebenen String in einen int zu konvertieren, kann die Methode `atoi(const char *str)` genutzt werden.

➔ `Stripe.setBrightness(atoi(bright.c_str()))`

Anschließend müssen die gesetzten werden wieder mit show() auf den „Hardware“-Stripe übertragen werden.

Um Die Werte über BLE in die entsprechende Charakteristik zu schreiben muss hier auch die Callback Methode `onWrite()` überschrieben werden. (TODO BrightnessCallback Methode). Dies funktioniert im selben Stil wie auch bei dem setzten der Farbewerte aus dem vorherigen Kapitel.

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.