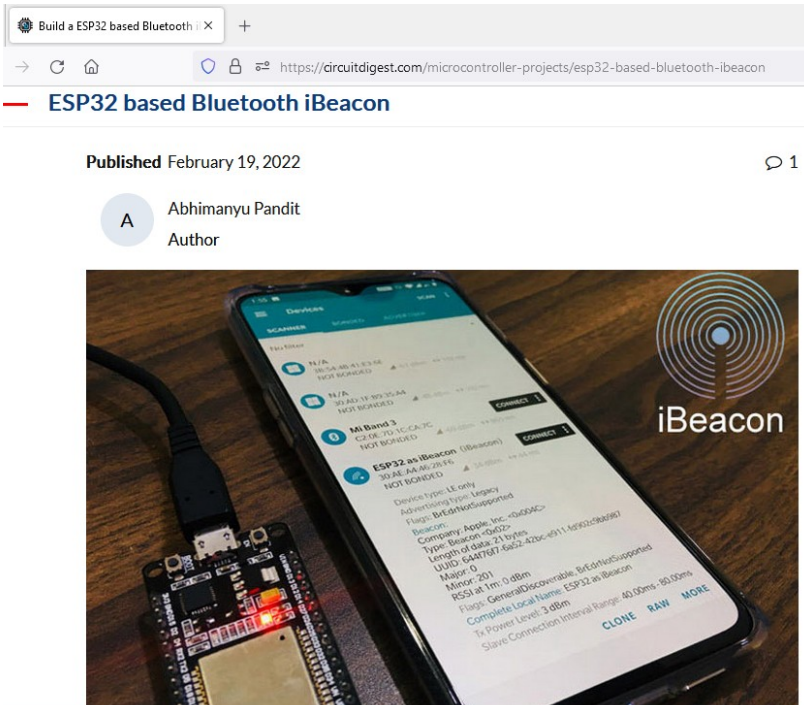
# Beacons: Selbstprogrammierbar mit ESP32, HM10-Modul oder Calliope

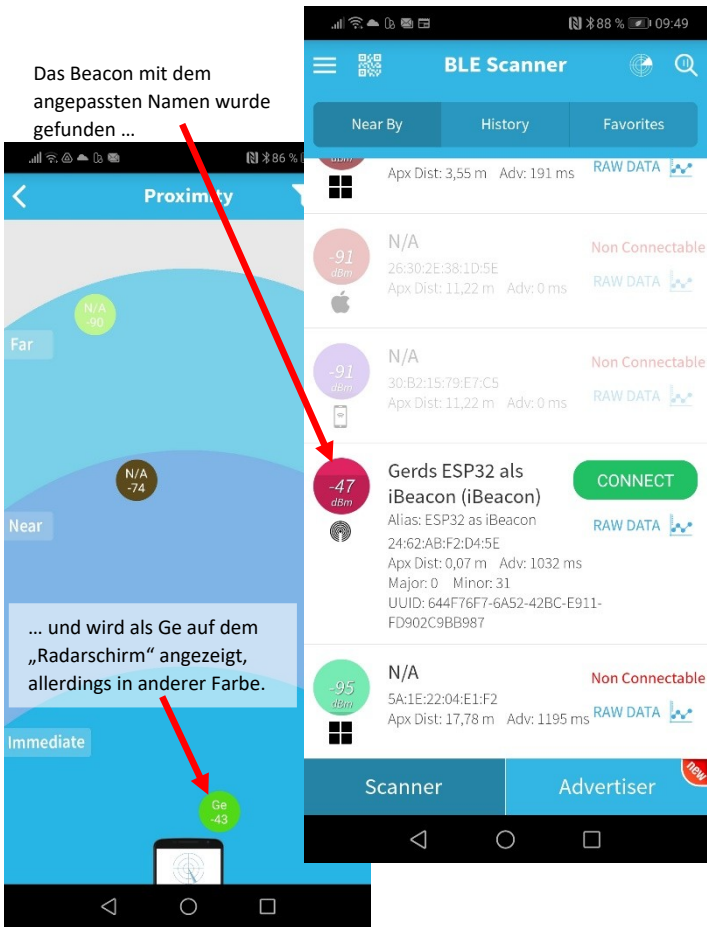



„Beacon“ kann mit „Signalfeuer“ oder „Funkboje“ übersetzt werden. Ein Beacon auf Bluetooth-Basis sendet immer wieder ein Funksignal mit Information aus. Diese erlauben es, ein Beacon mit einer Smartphone-App zu orten. Damit eignen Beacons sich als „Schlüsselfinder“. BLE als besonders energiesparende Bluetooth-Version erlaubt es, Beacons zu bauen, die mit einer Knopfzelle viele Monate funken. Wer in die BLE-Programmierung einsteigen will, sollte mit der Welt rund um Arduino und Co. Einigermaßen vertraut sein. Die Arduino-Entwicklungsumgebung ist gut geeignet, die entsprechende Hardware zu programmieren. Es werden allerdings keine Arduinos programmiert. Stattdessen kommen z.B. ESP32- oder HM10-Module zum Einsatz. Mit diesen beiden Varianten und zum Schluss mit dem Calliope beschäftigt sich dieser Text. Bei der Verwendung des ESP32-Boards habe ich mich eng an die Vorschläge von Herrn Pandit gehalten, die ich am 23.05.2023 auf [https://circuitdigest.com/microcontroller-projects/esp32-based-bluetooth-ibeacon](https://circuitdigest.com/microcontroller-projects/esp32-based-bluetooth-ibeacon) gefunden habe. Allerdings verwende ich nicht den von ihm vorgeschlagene „nRF Connect“-App, sondern den BLE-Scanner von „Bluepixel Technologies“.

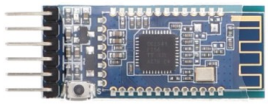

Den von Pandit vorgeschlagenen ESP32-Code habe ich an zwei Stellen verändert: Als Name des habe ich „Gerds ESP32 als iBeacon“eingetragen und bei „#define GPIO_DEEP_SLEEP_DURATION“ den Wert 1 (anstatt 10) gesetzt.

Der Code – im PDF auch kopierbar

```
#include "sys/time.h"
#include "BLEDevice.h"
#include "BLEUtils.h"
#include "BLEServer.h"
#include "BLEBeacon.h"
#include "esp_sleep.h"
#define GPIO_DEEP_SLEEP_DURATION     1  // sleep x seconds and then wake up
RTC_DATA_ATTR static time_t last;      // remember last boot in RTC Memory
RTC_DATA_ATTR static uint32_t bootcount; // remember number of boots in RTC Memory
// See the following for generating UUIDs:
// https://www.uuidgenerator.net/
BLEAdvertising *pAdvertising;   // BLE Advertisement type
struct timeval now;
#define BEACON_UUID "87b99b2c-90fd-11e9-bc42-526af7764f64"
// UUID 1 128-Bit (may use linux tool uuidgen or random numbers via https://www.uuidgenerator.net/)
void setBeacon() {
  BLEBeacon oBeacon = BLEBeacon();
  oBeacon.setManufacturerId(0x4C00); // fake Apple 0x004C LSB (ENDIAN_CHANGE_U16!)
  oBeacon.setProximityUUID(BLEUUID(BEACON_UUID));
  oBeacon.setMajor((bootcount & 0xFFFF0000) >> 16);
  oBeacon.setMinor(bootcount & 0xFFFF);
  BLEAdvertisementData oAdvertisementData = BLEAdvertisementData();
  BLEAdvertisementData oScanResponseData = BLEAdvertisementData();
  oAdvertisementData.setFlags(0x04); // BR_EDR_NOT_SUPPORTED 0x04
  std::string strServiceData = "";
  strServiceData += (char)26;    // Len
  strServiceData += (char)0xFF;  // Type
  strServiceData += oBeacon.getData();
  oAdvertisementData.addData(strServiceData);
  pAdvertising->setAdvertisementData(oAdvertisementData);
  pAdvertising->setScanResponseData(oScanResponseData);
}
void setup() {
  Serial.begin(115200);
  gettimeofday(&now, NULL);
  Serial.printf("start ESP32 %d\n", bootcount++);
  Serial.printf("deep sleep (%lds since last reset, %lds since last boot)\n", now.tv_sec, now.tv_sec - last);
  last = now.tv_sec;
  // Create the BLE Device
  BLEDevice::init("Gerds ESP32 als iBeacon");
  // Create the BLE Server
  BLEServer *pServer = BLEDevice::createServer();
// <-- no longer required to instantiate BLEServer, less flash and ram usage
  pAdvertising = BLEDevice::getAdvertising();
  BLEDevice::startAdvertising();
  setBeacon();
  // Start advertising
  pAdvertising->start();
  Serial.println("Advertizing started...");
  delay(100);
  pAdvertising->stop();
  Serial.printf("enter deep sleep\n");
  esp_deep_sleep(1000000LL * GPIO_DEEP_SLEEP_DURATION);
  Serial.printf("in deep sleep\n");
}
void loop() {}
```



Wenig tatsächlichen Aufwand, aber eine Menge Übung mit dem Material dürfte folgende Version erfordern, die z.B. mit dem 23. Mai 2023 auf der Seite von Funduino angebotenen Bauteil gelingen sollte. Von uns wurde sie bisher allerdings noch nicht im Detail ausprobiert:



Bluetooth BLE Modul - CC2541, vergleichbar mit HM10, iBeacon

★★★★☆ (2)

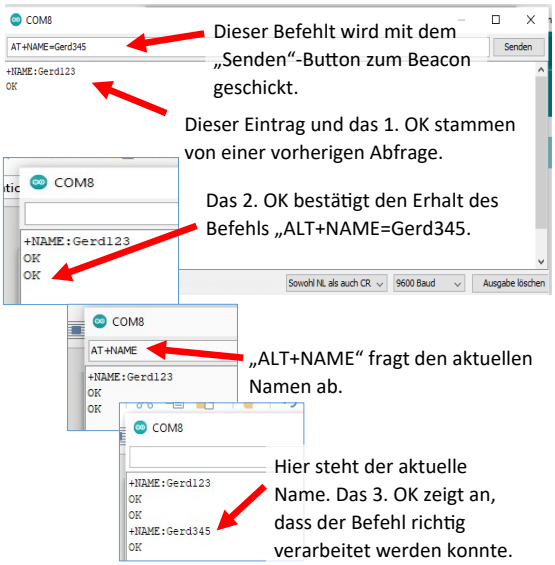
3,90 € *

Inhalt: 1 Stück
inkl. MwSt. zzgl. Versandkosten

Sofort verfügbar

1 Hinzufügen


Dieses Bauteil muss über einen sogenannten USB-TTL-Konverter (Schnittstellenwandler) an einen Rechner mit Arduino-Entwicklungsumgebung angeschlossen werden, so wie z.B. auf dem Bild unten links aus dem Video https://www.youtube.com/watch?v=ez3491-v8Og gezeigt wird.
Der rote Konverter ist über USB an den Rechner angeschlossen. In der Arduino-Enwicklungsumgebung muss der passende Port eingestellt werden, im Beispiel war es COM8. Als Übertragungsmodus muss immer „Sowohl NL als auch CR“ und als Geschwindigkeit 9600 Baud gewählt werden. Mit dem Senden des Befehl „AT+NAME=Gerd123“ wurde im Beispiel der Name des Beacons in „Gerd345“ umgewandelt und kann so vom Scanner erkannt und geortet werden.
(Die Abfolge der Meldugen im unteren Monitorfenster – siehe Bildfolge rechts - ist etwas gewöhnungsbedürftig. Die neuesten Medungen werden unten angesetzt. Und dass eine Umbenennung erfolgreich war, wird nicht mir der Meldung des neuen Namens quittiert. Dieser muss extra abgefragt werden.)


Calliopes sind teuerer als ESP32 oder HM10-Module, sind aber an vielen Schulen verfügbar und erlauben ebenfalls ein Kennenlernen der Beacon-Welt. Allerdings sind muss dazu eine Erweiterung von https://amerlander.github.io/pxt-bluetooth-beacons/ geladen werden, hier der QR-Code:
Sobald der Calliope entsprechend programmiert ist, wird er als Beacon im BLE-Scanner angezeigt und ermöglicht direkt aus ihm heraus die eingetragene Seite zu öffnen.