LWSC WiFi/LoRa Funk System Dokumentation

## Motivation

Der Verein „Lauterbacher Wurfscheiben Club e.V.“ ist ein Verein innerhalb des „Sportschützenverein e.V.“. Er befasst sich mit dem Flintenschießen auf Tontauben. Diese Tontauben werden von unterschiedlichen Maschinen in unterschiedlichen Richtungen auf einem ca. 200x500m großen Areal von unterschiedlichen Standpunkten geworfen und der Schütze muss versuchen, von seinem Standpunkt aus diese zu treffen.

Um ein unabhängiges und zentrales Auslösen dieser Maschinen zu ermöglichen, verfügen alle über ein Funkempfänger, der den Auslösemotor oder das Auslöserelais betätigt.

Das bisher eingesetzt Funksystem basiert auf einem analogen 433MHz System und wurde nicht mehr gewartet und war zunehmend fehleranfällig, weshalb ein neues System entwickelt werden sollte, das bei Fertigstellung das alte System ablösen sollte. Das neue System müsste skalierbar und einfach zu warten sein, außerdem sollte es unabhängig der eigentlichen Maschinen funktionieren und die Möglichkeit besitzen, in Zukunft weitere Funktionalitäten (wie Sensorik o.ä.) hinzuzufügen.

## Grundkonzept

Das Grundkonzept des neuen Funksystems baut auf den Espressif ESP8266 Chip auf. Dieser ist ein low-cost WiFi Mikrocontroller, welcher frei programmierbar ist.

Datenblatt ESP8266:

\*Prozessor: L106 32-bit RISC Mikroprozessorkern 80 MHz

\*Memory:

\*\*32 KiB Befehlsspeicher

\*\*32 KiB Befehlsspeichercache

\*\*80 KiB Benutzerdaten RAM

\*\*16 KiB ETS Systemdaten RAM

\*Externer Quad-SPI Flashspeicher: bis zu 16 MiB werden unterstützt

\*IEEE 802.11 b/g/n Wi-Fi

\*802.11n bis 72,2 Mb/s

\*\*Integrierte(r) T/R switch, balun, LNA, power amplifier und Leistungsanpassung

\*\*WEP oder WPA/WPA2-Verschlüsselung, ebenso offene Netzwerke

\*16 GPIO PINs

\*SPI

\*I²C (nur per Software implementiert)[6]

\*I²S Schnittstellen mit DMA (PINs mit GPIO geteilt)

\*UART auf einem dediziertem PIN, zusätzlich ein reiner Nur-Sende-UART, aktivierbar auf GPIO2

\*10-bit ADC mit sukzessive Approximation

\*RTC auf GPIO16, hiermit kann per Brücke zu RST der Chip aus dem Deep Sleep aufgeweckt werden

Durch eine große Community und einen Stückpreis von unter 2€ fiel die Wahl auf deinen Chip als „Herz“ des Empfängermoduls. Der Esp8266 kann sowohl als WiFi-Accesspoint als auch als Client agieren (auch beides gleichzeitig). Dabei gibt es jedoch die Beschränkung, dass nur max. 4 Clients von einem ESP8266 akzeptiert werden können, da die beschränkten Ressourcen nicht mit denen eines vollwertigen Accesspoints gleich kommen.

Diese Beschränkung spielt aber bei den Empfängermodulen keine Rolle, da kein managed WiFi genutzt wird, sondern ein propertäres Mesh-Netzwerk auf der Netzwerkschicht aufgebaut wird. Dies ermöglicht ein einfaches An/Ausschalten von Maschinen, ohne dass das Netzwerk als solches dann Verwaltungsaufwand wie Routen bekommt.

Auslöser/Sender sind ebenfalls ESP8266, welche das gleiche propertäres System zur Übertragung der Daten nutzen. Diese können entweder Befehle aus einem vorhandenen, klassischen WLAN „übersetzten“ oder direkt per Hardware-Pins starten.

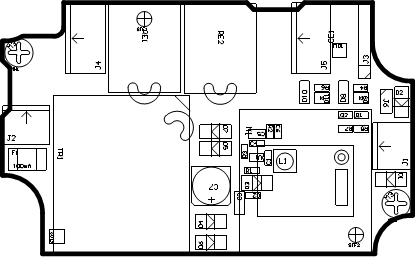
## PCB Empfänger

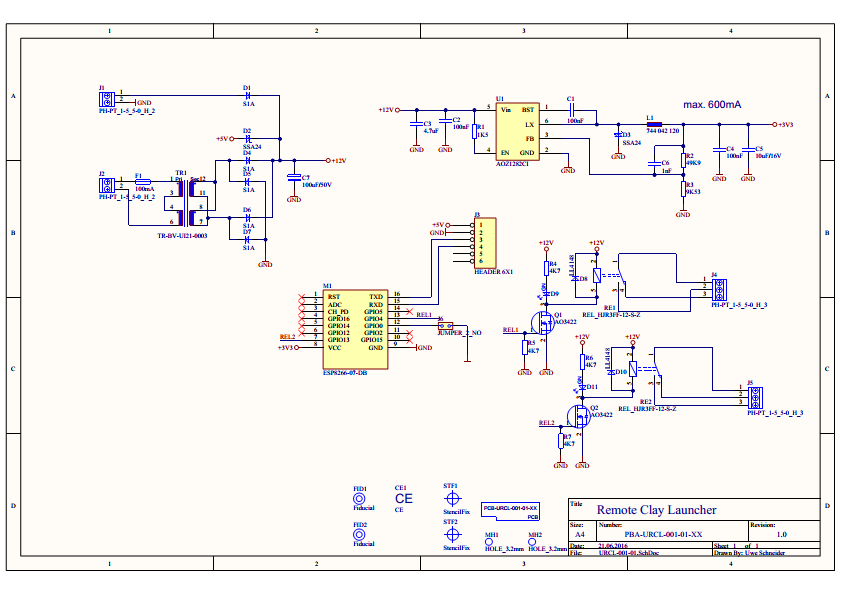
Die Schaltung des Empfängers und ihr Platinenlayout wurden selbst entworfen, und von der Firma „Uwe Schneider GmbH“ gefertigt und SMD Bestückt, die Endbestückung erfolgte durch eigene Hand.

Uwe Schneider GmbH  
Heegwaldring 12  
63694 Limeshain   
Tel.: +49 / 6047 / 98 65 21  
FAX.: +49 / 6047 / 98 73 96  
E-Mail: Gerald.Sander@UweSchneider.de

Dabei wurden 30 Platinen angefertigt. Sie verfügen über einen 12V und einen 230V Eingang zu Spannungsversorgung, ein Spannungswandler für die benötigten 3,3V für den ESP8266 sowie zwei 12V Relais zum Schalten der Ausgänge.

*Die folgenden Bilder sind gleichzeitig PDF-Dateien, Doppelklick öffnet diese:*





## Software

Die Software, mit der die ESP8266 geflasht werden, befinden sich in einem Git-Repository:

[https://github.com/thetemplar/EspOTAMesh/](https://github.com/thetemplar/EspOTAMesh/blob/master/EspOTAMesh.ino)

Es wird kein klassisches managed WLAN genutzt, um die Daten zu übertragen, bei dem jeder Teilnehmer eine IP bekäme. Die API von Espressif ermöglicht es, unter anderem eigene „Beacon Frames“ zu senden. Diese sind eigentlich dafür vorgesehen, dass Accesspoints möglichen Teilnehmern über ihre Existenz informiert, so wie Daten über sie und das Netzwerk, welchem sie angehören. So sendet ein normaler Accesspoint mehrmals pro Sekunde seinen Namen an alle Empfänger in Reichweite, damit diese sich auf ihn verbinden können. Nach der IEEE 802.11 Spezifikation gibt es in diesen Paketen die Möglichkeit eigene „Vendor Specific Information“ unterzubringen.

 *Link zur PDF der IEEE Spiezifikation über Beacon Frames (Doppelklick).*

Die Kommunikation der ESP8266 untereinander basiert nun auf solchen Beacon Frame Paketen, welche zwar zweckentfremdend sind, aber noch immer IEE 802.11 konform. Dabei wird die SSID leer gelassen, was einem „hidden“-WiFi entspricht und die Destination, Source und BSSID-Adresse abgeändert. Diese spiegeln eigentlich die MAC Adressen der jeweiligen Teilnehmer dar, werden aber durch die Chipadressen der ESP8266 ersetzt (und sind somit MAC equivalent). Alle minimal nötigen Anforderungen an ein Beacon Frame werden IEEE-konform erfüllt, plus die zusätzliche Vendor Specific Information.

Ein Minimalpaket des Meshes sieht dann aus wie folgt:

uint8\_t beacon\_raw[] = {

0x80, 0x00, // 0-1: Frame Control

0x00, 0x00, // 2-3: Duration

0xef, 0x50, 0xff, 0xff, 0xff, 0xff, // 4-9: Destination address (broadcast)

0xef, 0x50, 0x00, 0x00, 0x00, 0x00, // 10-15: Source address

0xef, 0x50, 0x00, 0x00, 0x00, 0x00, // 16-21: BSSID (transmitter)

0x00, 0x00, // 22-23: Sequence / fragment number

0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, // 24-31: Timestamp

0x64, 0x00, // 32-33: Beacon interval

0x31, 0x04, // 34-35: Capability info

0x00, 0x00, // 36-37: SSID (hidden)

0xDD, 0x05, 0x07, 0xFA, VERSION, START\_TTL, MSG\_Unknown //38-44++ Vendor Specific

};

Ziel/Source Adressen basieren auf der Chipid (4 byte) und zwei festgelegten Bytes davor: 0xef 0x50. Dabei bedeutet die Chip ID 0xffffffff ein Broadcast an alle Teilnehmer.

Der eigentliche Inhalt der Übertragungen beginnt ab Byte 40: Nachdem durch 0xDD das Feld der Vendor Specific Information aufgemacht wird, das eine Länge des folgenden Bytes besitzt (in dem Beispiel oben 5 bytes), folgt der eigentliche Payload für die Übertragung. Ein eigener Header beginnt nun mit einem nicht mehr genutzten Byte, gefolgt von