Gadget Entwicklung

Gesehen habe ich die Dinger während des Aufwärmtrainings der Roten Raben, als Timo Lippuner dort noch Cheftrainer war. 2, 3 blinkende Dinger, auf die die Libera reagieren musste, bevor sie einen Ball abwehren musste. „Will haben, aber was ist das?“. Die Startpage-Suche war nicht sehr ergiebig, und so rückten sie in den Hinterkopf, bis Melanie Cina im 22. Trainertalk davon erzählte. Blaze Pods heißen die Dinger, leider unerschwinglich. Also reifte ziemlich schnell die Idee: Selber bauen. Hier ist die Geschichte.

Designkonzept

Die Bedienbarkeit über Mobilgerät (Tablet, Handy, Android oder iOS, PC) erfordert eigentlich zwangsläufig eine Browserlösung mit Javascript. Das hat zwar ein paar hässliche Implikationen, weil die meisten Browserhersteller inzwischen der Meinung sind, lokal auf dem Gerät gespeicherte Dateien sind gefährlich, die von Servern aber nicht.

Von der Hardware Seite war gefordert: Robustes Gehäuse, integrierte Stromversorgung, WiFi oder BLE, leichte Programmierbarkeit. Bei der Wahl des Steuergerätes wurde mir der ESP32 empfohlen, im Rückblick eine gute Wahl. Ich habe mich für das Developmentboard entschieden, das erfordert nicht so ganz gute Lötfähigkeiten. Die ESP32 haben etliche IO-Pins, sind über C++ programmierbar, es gibt etliche Bibliotheken, und ganz wichtig: Etliche Tutorials.

Hardwarekonzept

Als Gehäuse verwende ich formschöne HT-Rohr Doppelmuffen mit Deckeln aus dem Baumarkt, die in einem ansprechenden Dunkelgrau gehalten sind. Meine sind DN75, also 75mm Durchmesser, und etwa 110mm hoch. Für die Halterung der LEDs habe ich mit einer Lochsäge Scheiben aus einer alten Korklaminat-Bohle geschnitten, die noch bei uns im Keller rumlag. Gegenüber MDF-Platten hat die den Vorteil, nur 11mm dick zu sein. Mit einem 6mm Holzbohrer einen 9er-Kranz für die LEDs gebohrt und zentriert auf den einen Deckel geklebt. Um das zentrale Loch zu verdecken, wird das ganze noch mit einer Linsenkopfschraube und Mutter auf der Rückseite ergänzt. Der ESP32 kann auch Touch-Ereignisse reagieren, deswegen habe ich die Oberseite mit Alu-Folie beklebt. Mit der Linsenkopfschraube wird die elektrische Verbindung zur Oberseite hergestellt. Das war leider ein Detail, in dem das Internet mich auf die falsche Fährte gelockt hat. Alle Tutorials zeigen das mit einem kleinen Draht, prima, klapp super. Im Kleingedruckten habe ich dann später gefunden, wie das elektronisch funktioniert. Der Controller misst Kapazitätsänderungen. Leider funktioniert das mit dem von mir gewählten Aufbau nicht. Die Alufolie bildet keinen Kondensator, der sich durch die Anwesenheit eines Fingers ändert, sondern lediglich eine Antenne. Signal-to-Noise wird grottig. Da war die Folie aber schon drauf. Also habe ich die Linsenkopfschraube isoliert, sie ist die einzig sensitive Fläche, aber das war elektrisch OK.

Im Deckel ist umlaufend ein 1.5mm² Kabel, das später die Versorgungsspannung heranschafft. Die LEDs sind „fliegend“ verlötet, jeweils drei einer Farbe parallel. Meine Töchter konnte ich damit überraschen, dass ich die drei Anschlusslitzen+Touchleitung in einen Zopf flechten konnte. Der führt zu einer kleinen Lochrasterplatine, auf die drei Schalttransistoren mit Vorwiderständen und Lastwiderstände gelötet sind. Der ESP32 kann eine LED treiben, aber keine 3, deswegen musste das

hier entkoppelt werden. Die Transistoren sind stinknormale 2N2222. Eine 5-polige Buchsenleiste ermöglicht den steckbaren Anschluss an den ESP32.

Zur Stromversorgung habe ich eine DC-Einbaubuchse in die Zylinderwand der Doppelmuffe eingebaut. Vorgesehen ist ein 5V Netzteil, das etwa 1A liefern können sollte. Von der Buchse geht es auf eine Ladeplatine für 3,7V LiIon/LiPo Akkus. An die kommt auf der anderen Seite ein 18650 Li-Ion Akku mit 3.6V/2600mAh. Die letzten beiden Anschlüsse versorgen die restliche Schaltung mit Spannung. Das tun sie auch, während geladen wird. Wer in der Lage ist, SMD-Widerstände aus- und wieder einzulöten, kann den Ladestrom begrenzen. Netzteile, mit denen man 5V/4A abrufen kann, sind dünn gesät, Deswegen sind 250mA eigentlich schöner.

Fast fertig, einen Schönheitsfehler hat das Ganze noch. Der ESP32 kann mit 5V oder 3.3V versorgt werden. Bei den 5V ist er so semi-tolerant, bei den 3.3V eher pingelig. Deswegen wird das ganze noch mit einer weiteren Mini-Platine ergänzt. Auf der sind ein LDO MCP1700-3302E, ein 1000µF Elko und ein 100nF Keramikkondensator verbaut. Der MCP ist ein Low-Drop-Out Spannungsregler, der 3.3V liefert. Der Elko stabilisiert die Spannung, wenn der ESP32 gerade im WiFi funkt, da zieht er gerne mal über 200mA, der 100nF dämpft HF-Oszillationen. Komplettiert wird der ganze Aufbau durch einen einfachen Schalter neben der DC-Buchse, die die 3.3V auf den ESP durchschaltet.

Zum Einbau werden die Platine mit Isolierband umwickelt (ich hab vergessen Schrumpfschlauch zu bestellen 😕) und einigermaßen sicher mit Verpackungschips aufgefüllt. Wie robust das im Trainingsbetrieb ist, wird sich erst nach Corona zeigen.

Software auf dem ESP32

Als Entwicklungsumgebung habe ich Arduino mit der ESP-Erweiterung unter Linux genutzt. Im wesentlichen hat die Firmware auf den ESPs drei Funktionen:

1. Kontakt mit einem WLAN aufnehmen
2. Updates über die OTA entgegennehmen
3. Kommunikation mit den Clients und Management der LEDs und des Touch-Eingangs

Kontakt mit einem WLAN aufnehmen

Updates über OTA

Eine großartige Funktion der ESPs, die leider erst spät entdeckt habe, ist das Update über WiFi. Den ESP einmal richtig einbauen, WLAN konfigurieren und gut. Der realisierte Code ist direkt Copy&Paste aus dem Internet. Es gibt noch eine Version, die besser auf den verwendeten Web-Server zugeschnitten ist, aber da war am Anfang nicht ganz klar, welche Teile für den kleinen Bruder (8266) und welcher für die ESP32 gilt. Wenn ich mal Zeit habe…

Kommunikation mit den Clients

Die Kommunikation erfolgt auf zwei Ebenen. Zum einen steht ein WebServer bereit, der statische

Seiten serviert. Die Ursprungsidee war, dort nur die Konfiguration abzuhandeln und die eigentlichen LED-Programme lokal auf den Clients zu speichern. Das funktioniert auch prima mit dem FireFox auf dem Desktop, aber so gar nicht mit mobilen Clients unter Android (Safari unter iOS habe ich dann gar nicht mehr probiert...). Dort sind die Browserhersteller der Meinung, lokale Dateien sind viel gefährlicher als Web-Seiten, deswegen muss das unbedingt verboten sein. Kann man sich gar nicht ausdenken, so was.

Also braucht man entweder einen externen Web-Server oder die Seiten müssen auf den ESP. Wenn man als AP einen RasPi 4 nimmt, kann man auch noch gleich einen nginx oder lighttpd draufsetzen und gut. Das verkompliziert aber den Aufbau, also kommen die Seiten auf die Micros. Da der Speicher dort recht knapp ist, wurden die Seiten vor dem Upload weitgehend komprimiert. Einmal durch einen Minifier, und dann mit gzip verkleinert, bevor sie im PROGMEM der Micros landen. Beim Ausliefern muss lediglich noch der Header für „Inhalt kommt komprimiert“ gesetzt werden, das Entpacken übernimmt der Client auf dem PC/Smartphone.

Es gibt ein paar spezielle Seiten. Unter der Adresse „http://esp/scan“

- Async WebServer
- Start mit AP&Versuch, in ein WLAN einzuloggen. Grün Blink: Versuch
- Wenn in WLAN, Stop AP
- Ansonsten bleibt AP 20 Minuten aktiv, dann Schlaf
- in AP: http://192.168.4.1/scan mit reload, set name, SSID&PW

Web-Sockets, clr, CLR set SET, STATE

Software auf dem Browser

Anwendung in der Praxis