HA Personal | Sven Mathäs
Ausbildung, Fortbildung und Personalentwicklung
Auszubildender Informationselektronik G/S

Mainz, 06.02.2022

Projektarbeit

Mehrfach-Tally Übertragung über Ethernet
realisieren und implementieren

E-Mail | mathaes.s@zdf.de

Inhaltsverzeichnis

Vorwort

Bereits Mitte 2021 habe ich mir Gedanken über mein Gesellenstück gemacht. Mir sind einige Ideen eingefallen, jedoch wollte ich ein Projekt erarbeiten, das auch wirklich gebraucht wird und nicht im Regal Staub fängt. Auf der Suche nach geeigneten Projekten habe ich verschiedene Teams befragt. Die Idee, die schließlich meine Projektarbeit geworden ist, kam aus dem Team „Drahtlos Bild". Das Projekt soll ein Übertragungssystem für Tally-Signale über Netzwerk werden.

Einführung

Mein Ausbildungsbetrieb (ZDF)

Mein Ausbildungsbetrieb, das Zweite Deutsche Fernsehen, ist ein öffentlich-rechtlicher Fernsehsender mit dem Hauptsitz in Mainz-Lerchenberg. Gemeinsam mit der ARD und dem Deutschlandradio bildet mein Ausbildungsbetrieb den öffentlich-rechtlichen Rundfunk in Deutschland und gehört zu den größten öffentlich-rechtlichen Sendeanstalten in Europa.

Rotlicht oder Tally

Unter dem englischen Fachbegriff Tally versteht man ein Aufnahmelicht für die Signalisierung von der Benutzung einer Signalquelle oder eines Studios[1]. Das Tally ist dabei ein kleines Rotlicht an den Fernsehkameras, den Monitoren, den Mikrofonen und ggf. den Türen. Dieses zeigt Personen im Studio / Set an, welche Signalquelle aufgezeichnet oder ausgestrahlt wird.[2] Dementsprechend leuchtet für gewöhnlich immer nur bei einer Kamera ein Tally. Zudem wird auf einem Monitor mit einer Bildteilung, eine rote Umrandung um die angewählte Quelle eingeblendet.[3]

Dabei soll ein Tally:

- signalisieren, welche Kamera „auf Sendung" ist
- ein Mikrofon als eingeschaltet kennzeichnen
- irrtümliche Bedienung der Geräte verhindern
- dem Kamerastab anzeigen, dass sie „auf Sendung" sind und sich auf gesondertes Verhalten einstellen

Ein Tally-Controller übernimmt dabei die Steuerung mehrerer Tallys. Dieser Controller ist meist mit einem Bildmischer, Videomatrix (auch Kreuzschiene genannt) und anderen Peripheriegeräten verbunden. Wenn ein Signal ausgewählt wird, schließt der Tally-Controller einen Kontakt und schaltet dadurch das zu dem Signal gehörende Tally.

Tallys können auch andere Farben haben; dazu zählen Gelb und Grün. Je nach Einsatz werden die Bedeutungen dieser definiert. In den meisten Sendungen des ZDF, die andere Tally-Farben als Rot benutzen, bedeutet Gelb eine Vorauswahl („Du kommst gleich dran“) und Grün, dass die Quelle auf z.B. einer Leinwand angezeigt wird und somit den Sendungsverlauf indirekt beeinflussen könnte.

In Abbildung 1 ist eine Kamera mit aktivierter Tally-Leuchte am Objektiv und am Viewfinder.

Abbildung 1

Ausgangslage

Unter anderem wickelt mein Unternehmen Außenübertragungen ab. Das heißt, eine Videoproduktion wird außerhalb unseres Standortes Mainz-Lerchenberg aufgezeichnet oder gesendet. Dazu besitzen wir Mobile Produktionseinheiten und Übertragungswagen, kurz MPE und Ü-Wagen. Mit diesen Einheiten können wir vor Ort Kamerasysteme mit vollem Funktionsumfang nutzen.

In Zukunft soll dieses Prinzip, vor allem für Produktionen außerhalb der EU, abgelöst werden. Es soll künftig ein Kernteam ohne großen Ü-Wagen vor Ort, mit wenigen Kodierern, Sendern und Steuerungsgeräten eingesetzt werden. Von Mainz aus wird die Technik & Kameras ferngesteuert. Ein aktuelles Beispiel wäre hier, die Olympischen & Paralympischen Spiele in Tokyo, bei denen wir heute viele Funktionen fernsteuern.

Das bisherige System, das für die Tally Übertragung genutzt wird, ist bereits einige Jahre alt. Das System überträgt bis zu 6 Tallys über ein Mehrfrequenzwahl-verfahren.

Dennoch ist dieses System sehr beliebt und gefragt bei den Kollegen. Oftmals sind alle Sender-/Empfänger-Kombinationen vergriffen. Es besteht ein Mangel an Geräten und ich möchte dies beheben.

Mein Projekt implementiert die Datenübertragung über das moderne Ethernet Protokoll. Somit können die Informationen über bestehende Netzwerk-verteilungen publiziert werden. Damit fallen zusätzliche Signalwege weg. Diese können für andere Ressourcen verwendet werden. Um mehr als einen Sender und somit mehr als 8 Tallys übertragen zu können, ist eine Gruppenzuteilung möglich. Die Fehlerdiagnose und Erweiterungsmöglichkeit sind durch ein Diagnose-Port an der Front-Blende und interne SPI-Steckleiste vereinfacht.

Ziel

Mein fertiges Gesellenstück soll aus einer Sender-/Empfänger-Kombination bestehen, basierend auf der Teensy4.1 Plattform. Anders als das alte System, ist mein System modular aufgebaut und folgende Funktionen sind enthalten:

- Übertragung der Statusdaten per Ethernet
- Einteilung in Gerätegruppen flexibel möglich (z.B. mehrere Systeme im gleichen Netz oder mehrere Empfänger pro Sender)
- 8 Tally Eingänge / Ausgänge pro Gerät über XLR
- Diagnose Port an der Frontblende
- Status-Anzeige für Einzelne Tallys
- Erweiterungsmöglichkeit über eine SPI Steckleiste
- Ausführliche technische Dokumentation für die Wartung von Hard- und Software

Hardware

Bauteilrecherche

Mein Projekt ist eine neue Interpretation zu dem bestehenden Tally-Sender und -Empfänger. Wie das bisherige System, befindet sich mein Projekt in einem 19 Zoll 1HE Gehäuse pro Gerätetyp. Die äußeren Anschlüsse befinden sich an den gleichen Positionen am Gehäuse.

µC Teensy 4.1

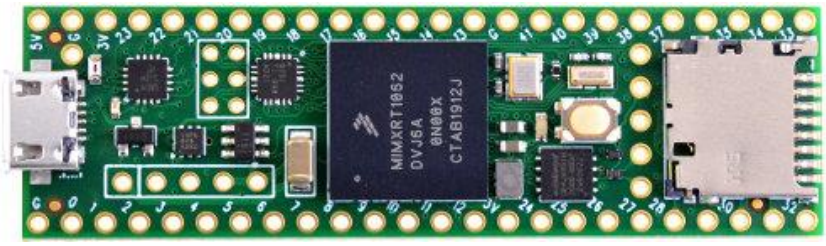

Abbildung 2 – der Teensy 4.1

Der Teensy ist ein komplettes auf USB basierendes Mikrocontroller-Entwicklungssystem mit einem sehr kleinen Fußabdruck, mit dem sich viele Arten von Projekten realisieren lassen. Jegliche Programmierung erfolgt über den USB-Anschluss.[4]

Besonderheiten des Teensy 4.1?

Mit einem ARM-32bit-Mikrocontroller getaktet auf 600MHz, 7936 KB Flash und 1024 KB Arbeitsspeicher ist der *Teensy 4.1* ein echtes Biest[4]. Der *Teensy 4.1* besitzt zudem eine native Ethernet-Schnittstelle; nur ein Anschluss muss hinzugefügt werden. Gerade durch diese Eigenschaften ist meine Auswahl auf dem *Teensy 4.1* gefallen.

Alternative Überlegung

Als Alternative Überlegung stand auch ein Mini-Computer wie der *NanoPi NEO CORE* [5] in der Auswahl. Dieser ähnelt einem *Raspberry Pi* ohne direkte Anschlüsse wie USB oder Netzwerk. Auf diesem wäre dann ein Linux-System gelaufen, das meine Skripte ausführt und sich um die Kommunikation im Hintergrund kümmert.

Warum ein Mikrokontroller?

Ein Mikrocontroller ist schneller im Umgang mit Ein- & Ausgängen. Im Gegensatz zu einem Einplatinencomputer wie dem *Raspberry Pi* oder dem *NanoPi* ist der Stromverbrauch geringer. Dies ist notwendig im Fall eines Spannungsverlustes; USV-Geräte könnten dadurch länger laufen. Das wichtigste Argument für einen Mikrocontroller ist die verkleinerte Angriffsfläche. Ein Linux-Betriebssystem ist angreifbarer und muss daher öfter gewartet werden.

Schieberegister SN74HC595N

Für die Ansteuerung der LEDs habe ich das SN74HC595N von Texas Instruments gewählt. Das Schieberegister hat 8 Ausgänge und ist seriell ansprechbar. Das Spezielle: Dieses Schieberegister besitzt ein Speicherregister. Somit kann man die Werte im Schieberegister verändern, ohne dass das Schieben der Bits als flackern an den LEDs zu erkennen ist. Das Byte wird erst in das Speicherregister übernommen, nachdem ein Impuls an Pin 12 anliegt. [6]

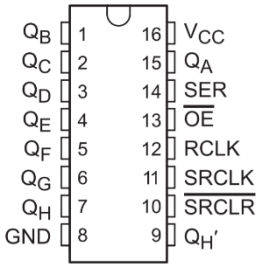

Abbildung 3

Doppel-Optokoppler HCPL-073L

Für eine galvanische Trennung zwischen meiner Eingangsschaltung und dem Teensy werden Optokoppler benötigt. Ich habe mich für den HCPL-073L von Broadcom entschieden. Dieser Optokoppler ist für den Einsatz in LVTTL/LVCMOS- oder anderen Low-Power-Anwendungen vorgesehen. Der HCPL-073L ist in einem SOIC-8-Gehäuse untergebracht. Die LEDs haben eine Vorwärtsspannung von 1,5 V bei einem Strom von 1,5 mA. [7]

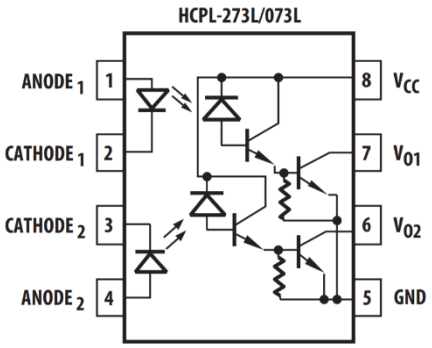

Abbildung 4 - Funktionsdiagramm

Relais-Treiber TPL9202

Der TPL9202 ist mein kleinster verwendeter SMD Relais-Treiber. Eine serielle Kommunikationsschnittstelle steuert die acht Low-Side-Ausgänge; jeder Ausgang verfügt über eine interne Snubber-Schaltung[9] zur Absorbtion der induktiven Last beim Ausschalten. Alternativ kann das System eine Fly-Back-Diode zu VIN verwenden, um die Rückführung der Energie in einer induktiven Last beim Ausschalten zu unterstützen.

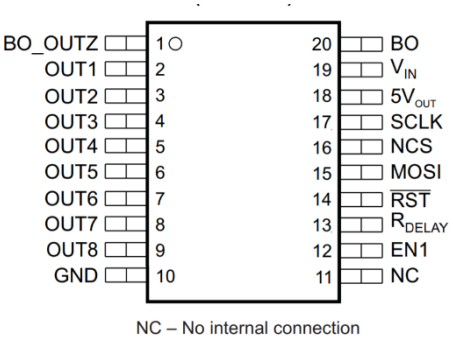

Abbildung 5 – TPL9202

Optional liefert ein Netzteil einen geregelten 5-V-Ausgang zur Versorgung eines Mikrocontrollers. Der Ausgang zur Erkennung von Spannungsabfällen (BO_OUTZ) warnt das System bei einem vorübergehenden Abfall der Versorgungsspannung, so dass das System potenziell gefährliche Situationen verhindern kann.
Die 5V Versorgung und BO_OUTZ habe ich nicht verwendet. [8]
In Kapitel [BUGFIXING, Seite 32] kommt dieser IC nochmals vor.

Spannungsisolatoren

Für den Schutz meiner Bauteile habe ich in der Spannungsversorgung isolierte DC-DC eingefügt. Diese sorgen für eine galvanische Trennung und gleichen kleine Schwankungen der Versorgungsspannung aus.

Ich habe die Wandler SPUN02M-05[10] und MDS20A-12[11] von Mean Well benutzt. Der SPUN02M-05 wandelt die 12 V Versorgungsspannung in 5V für meinen Mikrocontoller und meine Eingangsschaltung. Der MDS20A-12 gibt eine 12 V Spannung aus und dient dadurch nur zu Trennung und Stabilisierung der Relais-Schaltung. Der SPUN02M-12 reicht nicht aus für das Betreiben mehrerer Relais.

Abbildung 6 – SPUN02M-05

Abbildung 7 – MDS20A-12

Anschlüsse Neutrik

Neutrik-Standard

Im ZDF werden mit einigen Ausnahmen zum Großteil Stecker und Buchsen von *Neutrik* verwendet, sodass diese mir zu Verfügung standen. Das Design der Buchsen vereinheitlicht verschiedene Anschlusstypen im gleichen Gehäuse. Die D Serie von Anschlüssen haben alle denselben Fußabdruck und benötigen daher die Bohrungen an gleichen Positionen. [12]

XLR 3pol weiblich

Der Standard-3pol XLR[13] Anschluss kann nicht nur für Tonsignale und DMX Steuerung benutzt werden. Die vorhandenen Tally Leuchten werden beispielsweise durch ein Schließen von Pin 2 & 3 geschaltet.

Abbildung 8 – 3pol XLR weiblich

XLR 4pol männlich

Die männliche 4pol XLR hat sich bei uns als Standard für 12V Spannungsversorgung von Kameras und selbstentwickelten SK3-Geräten durchgesetzt.
Der Hersteller Sony war einer der Ersten, der diesen Anschluss für eine externe Spannungsversorgung an ihren Profi-Kameras verwendet.
Somit fiel die Wahl auf diesen Stecker. Die Belegung ist:

Abbildung 9 – 4pol XLR männlich

Pol	**1**	**2**	**3**	**4**
	Masse			Vcc (12V)

RJ45 Durchführung

Für den Anschluss an das Netzwerk habe ich mich für Neutriks Ethercon entschieden. Auf Außenübertragungen wird sehr oft auf- und abgebaut. Der Standard RJ45 Stecker ist einer von unseren empfindlichsten Steckverbindungen. Häufig werden an Netzwerkkabeln neue RJ45-Stecker gekrimpt. Der Vorteil: Durch die Ummantelung, ähnlich wie beim XLR-Stecker, ist der Stecker gesichert vor versehentlichen Auftreten oder gewaltvollem Ziehen. Zudem kann ein normaler RJ45-Stecker eingesteckt werden.

Abbildung 10 – RJ45 Durchführung

USB Durchführung

Auf der Frontblende habe ich zusätzlich einen USB-Anschluss gesetzt. Hiermit soll man später den Controller beschreiben, einstellen und debuggen können, ohne das Gerät öffnen zu müssen.

Abbildung 11 – USB Durchführung

Schaltplanerstellung

Für die Erstellung meiner Schaltpläne und Platinenlayouts benutze ich das Programm *KiCAD*. Dieses wird von einer Open-Source Community entwickelt und von CERN unterstützt. [14]

Abbildung 12 – KiCAD Logo

Aufgrund der Komplexität der Leiterbahnen in meinem Projekt, habe ich mich entschlossen einen Autorouter einzusetzen. Meine Wahl ist dabei auf *FreeRouting*[15] gefallen.

Sender

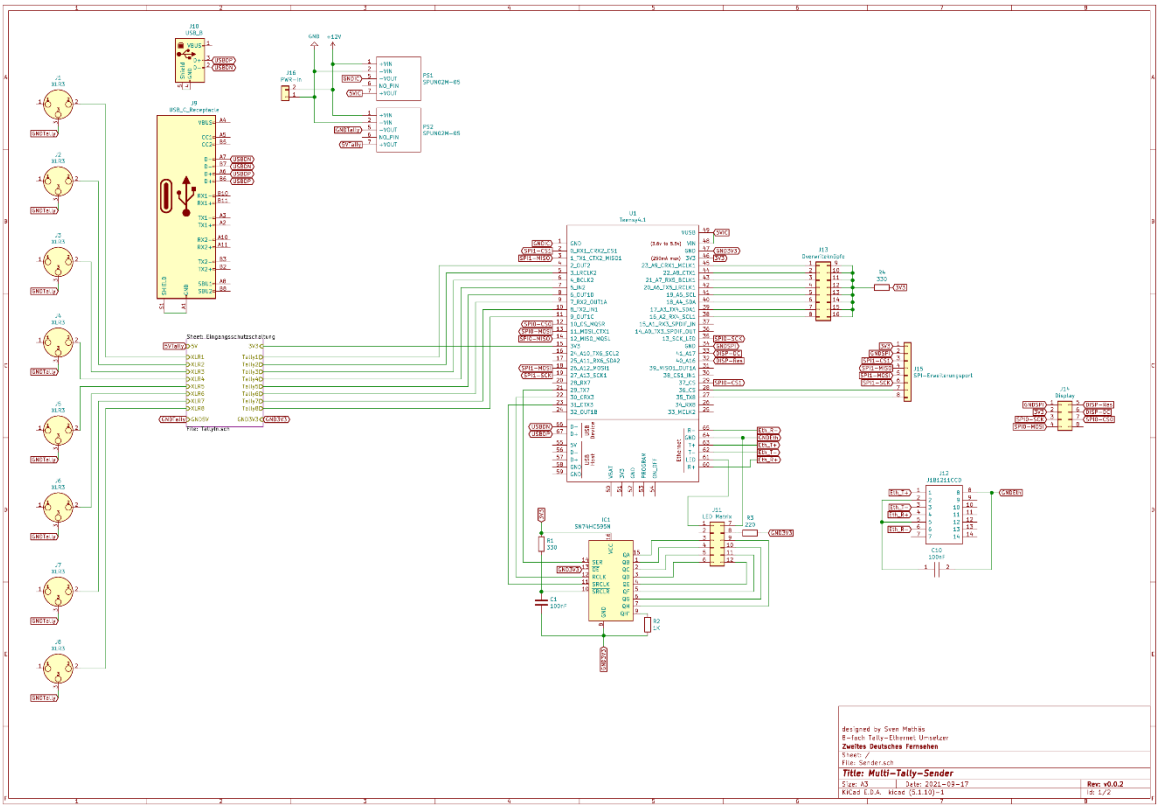

Abbildung 13 – Schaltplan Sender

Zur besseren Lesbarkeit ist dieser Schaltplan nochmals als DIN A3 Druck in den Anlagen zu finden.

Eingangsschutzschaltung

Auf die Erkennungsschaltung möchte ich näher eingehen.
Es existieren unzählige Standards von Tally-Steuerungen, die man in zwei Gruppen einteilen kann: Geschaltete Masse oder geschaltete Spannung.
Im ZDF wird die Masse mit schließen von Pin 2 & 3 an einem XLR-Stecker geschaltet. Um dies zuerkennen wird ein PNP-Transistor benutzt. [16]

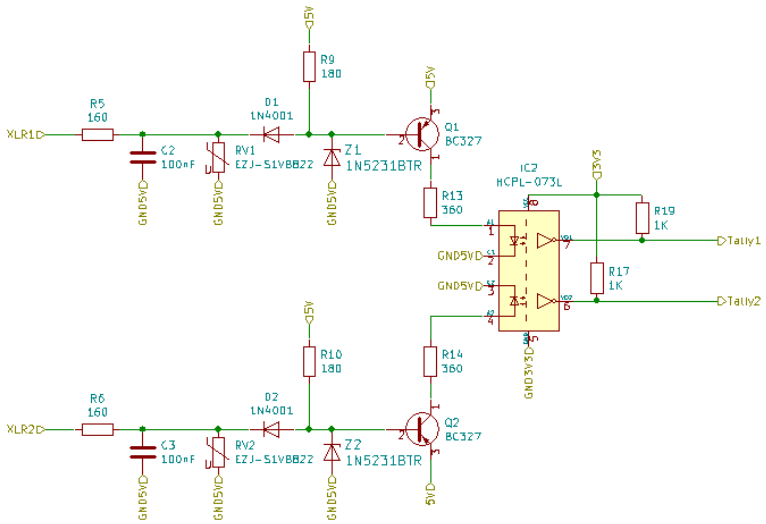

Abbildung 14 – Schaltplan Eingangsschaltung

Für den weiteren Schutz meiner Komponenten habe ich einen Varistor, eine Z-Diode und einen Optokoppler verbaut.

Empfänger

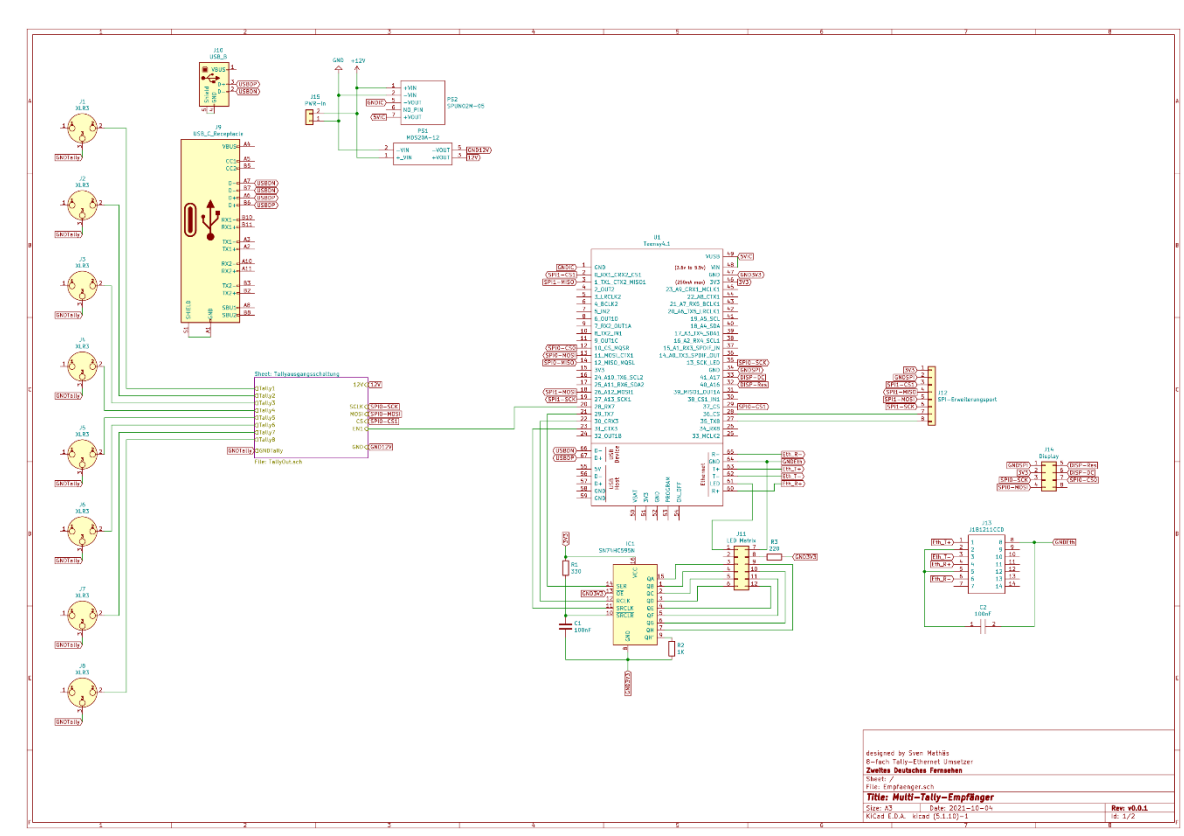

Abbildung 15 – Schaltplan Empfänger

Zur besseren Lesbarkeit ist dieser Schaltplan nochmals als DIN A3 Druck in den Anlagen zu finden.

Relais-Treiber

Den Relais-Treiber habe ich wie folgt eingeplant. Auch, wenn der Treiber-IC eine Dämpfungseinheit (Snubber) besitzt, habe ich zur Sicherheit Freilaufdioden eingefügt, um die hohe Abschaltspannung zurückzuführen. Dieser IC kommt erneut in Kapitel [BUGFIXING, Seite 32] vor. [16]

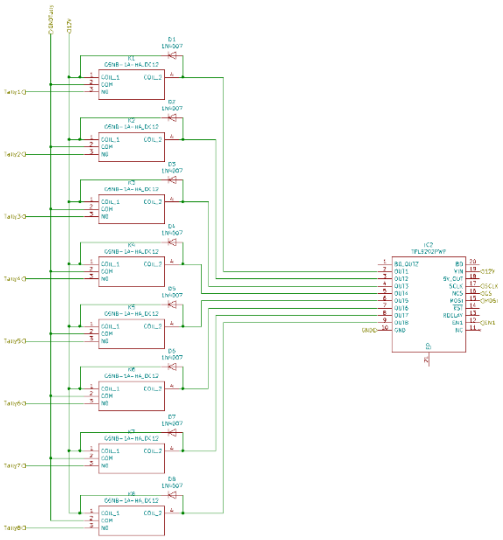

Abbildung 16 – Schaltplan Relais

Platinendesign

Für das Platinendesign habe ich KiCAD gewählt und danach mit FreeRouting geroutet. Die benötigten Platinen habe ich bei *Aisler*[17] bestellt. Entschieden habe ich mich für zweiseitige Platinen.

Sender

Die Bauteile habe ich in Gruppen angeordnet, sodass diese gut zu unterscheiden sind. Die erste Gruppe besteht aus den Spannungsisolatoren. Die zweite Gruppe sind die Eingangsschaltungen mit Optokopplern. Und die dritte Gruppe der Mikrocontroller und weitere Peripherie.

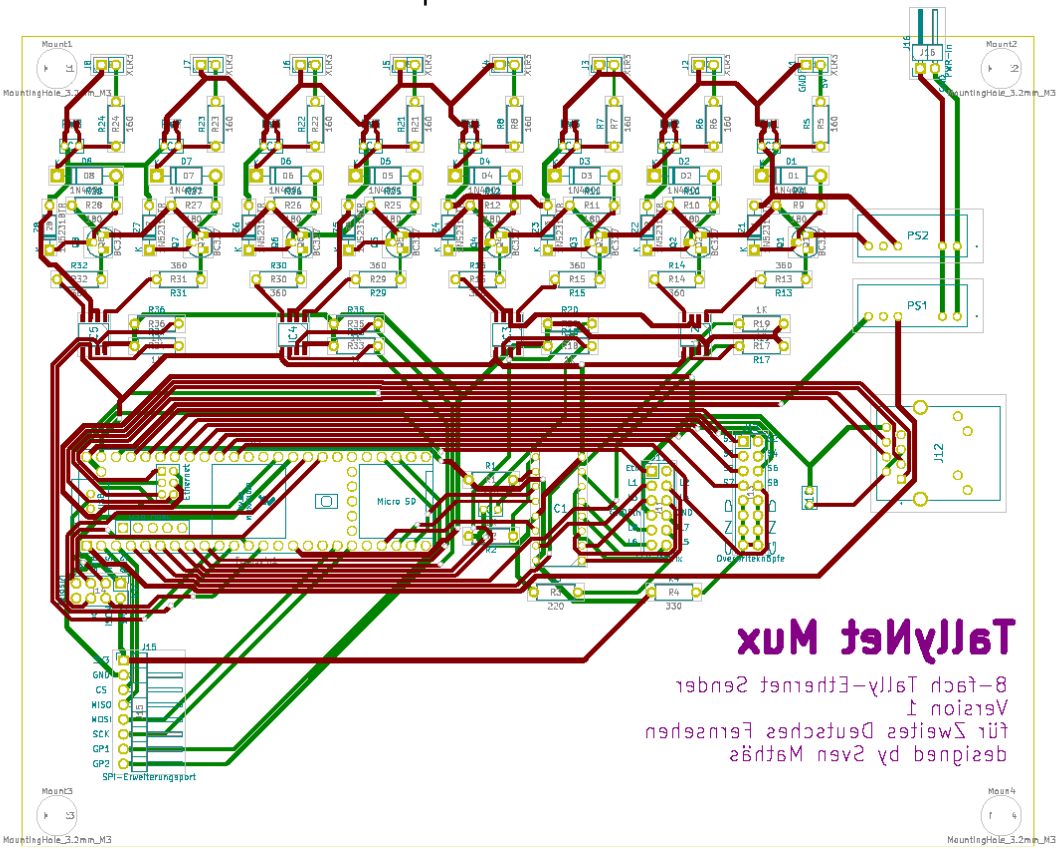

Abbildung 17 – Layout Sender

Zur besseren Lesbarkeit ist dieser Schaltplan nochmals als DIN A3 Druck in den Anlagen zu finden.

Empfänger

Empfänger und Sender unterscheiden sich nur in einigen Punkten. In der ersten Gruppe wurde ein Spannungsisolator durch einem 12V Isolator ersetzt. Die zweite Gruppe wurde komplett ausgetauscht mit den Relais und Treibern. Zudem findet man in der dritten Gruppe keine Steckleiste für Taster.

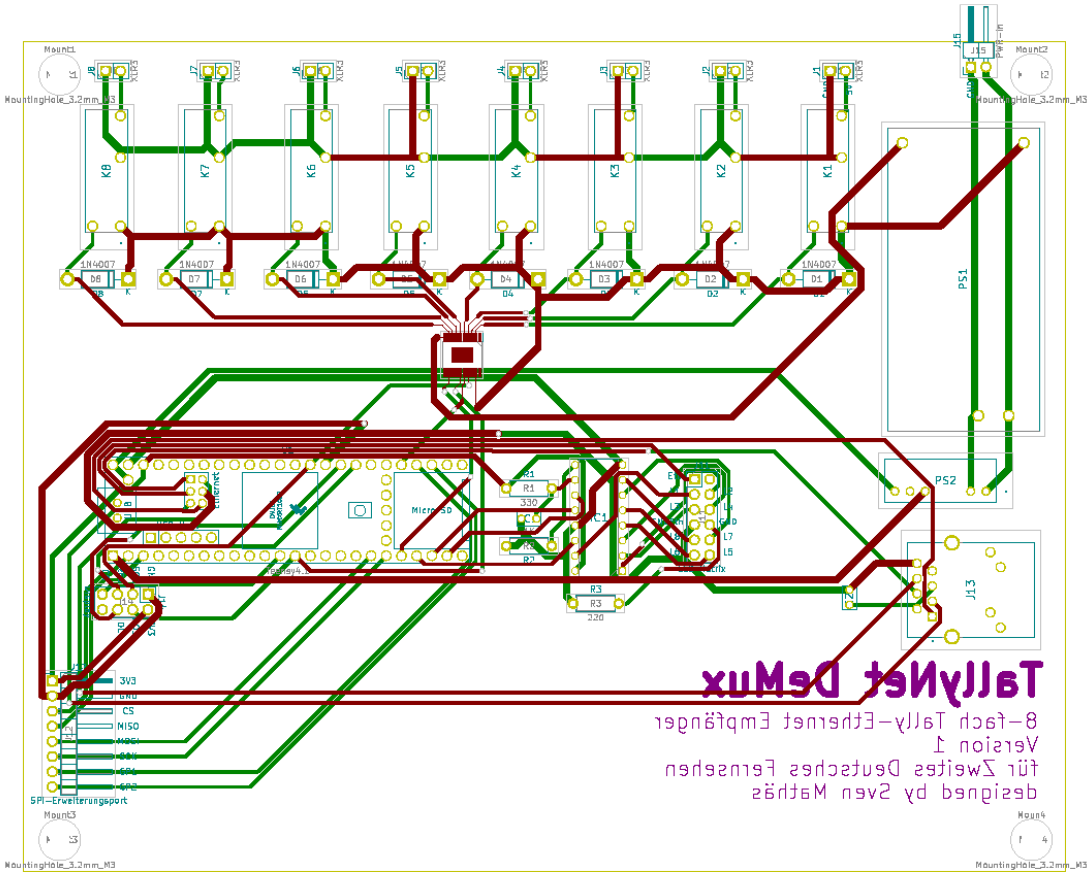

Abbildung 18 – Layout Empfänger

Zur besseren Lesbarkeit ist dieser Schaltplan nochmals als DIN A3 Druck in den Anlagen zu finden.

Material- und Bestückungsliste

Ref_Tx	Ref_Rx	Value	Menge	Vendor	Bestellnummer
S1 - S8		Drucktaster	9	Adafruit	1439
G1	G1	Display	3	Adafruit	2719
C1 - C10	C1, C2	100nF	15	Mouser	594-K104K15X7RF5TL2
D1 - D8		1N4001	10	Mouser	625-1N4001-E3/54
IC1	IC1	SN74HC595N	3	Mouser	595-SN74HC595N
IC2 - IC5		HCPL-073L	6	Mouser	630-HCPL-073L-000E
J1 - J8	J1 - J8	Neutrik Einbau XLR 3-pol female	16	Mouser	568-NC3FDM3LBAG-1
J10	J10	USB_B	3	Mouser	568-NAUSB-W
J12	J13	Ethernet Magjack	3	Mouser	950-J1B1211CCD
PS1 - PS2	PS2	SPUN02M-05	5	Mouser	709-SPUN02M-05
Q1 - Q8		BC327 PNP	10	Mouser	833-BC327-25-AP
R1,R4	R1	330	4	Mouser	279-ROX05SJ330R
R2, R17 - R20, R33 - R36	R2	1K	11	Mouser	279-ROX05SJ1K0
R3	R3	220	3	Mouser	279-ROX05SJ220R
R5 - R8, R21 - R23		160	10	Mouser	279-ROX05SJ160R
R9 - R12, R25 - R28		180	10	Mouser	279-ROX05SJ180R
R13 - R16, R29 - R32		360	10	Mouser	279-ROX05SJ360R
RV1 - RV8		EZJ-S1VB822	10	Mouser	667-EZJ-S1VB822
Z1 - Z8		1N5231BTR	10	Mouser	78-1N5231B
J99	J99	Neutrik Einbau XLR 4-pol male	2	Mouser	568-NC4MD-L-BAG-1
J88	J88	Neutrik Ethercon	2	Mouser	568-NE8FDX-P6
	IC2	TPL9202PWP	2	Mouser	595-TPL9202PWP
	K1 - K8	G5NB-1A-HA_DC5	9	Mouser	653-G5NB-1A-HADC5
	PS1	MDS20A-12	2	Mouser	709-MDS20A-12
D9	D1 - D9	LED für Status Anzeige	11	Mouser	630-HLMP-LG71-XZ0DD
		Jumper Kabel male-male	1	Mouser	485-4482
		Jumper Kabel female-female	1	Mouser	485-4447
J11	J11	Stiftleiste 2.54mm für LED Matrix	4	Reichelt	MPE 087-1-006
J13		Stiftleiste 2.54mm für Overwriteknoepfe	6	Reichelt	MPE 087-2-008
J15	J12	Stiftleiste 2.54mm für SPI-Erweiterungsport 90°	2	Reichelt	BKL 10120185
U1	U1	Teensy4.1	2	Reichelt	TEENSY 4.1
J97 - J89	J97 - J89	Stiftleiste 2.54mm für XLR	18	Reichelt	MPE 087-1-002
Gehäuse	Gehäuse	1HE 19" Cassis	2	Reichelt	SIN 1 HE SW
		Stiftleiste 2.0mm für Ethernet	3	Reichelt	MPE 150-3-006
		Buchsenleiste für Ethernet	3	Reichelt	MPE 156-3-006
		Buchsenleiste für Teensy	8	Reichelt	BKL 10120951
		Stiftleiste 2.54mm für Teensy	5	Reichelt	MPE 087-1-024
J14	J14	Stiftleiste 2.54mm für Display	x		
J98	J98	Stiftleiste 2.54mm für Spannungsversorgung 90°	x		

Gehäusebearbeitung

Meine Wahl ist auf ein Universalgehäuse gefallen, dazu müssen für die Anschlüsse Aussparungen gebohrt werden. Für das Design meiner Front- und Rückseite habe ich *LibreCAD* benutzt. LibreCAD ist ein Open-Source Programm 2D-CAD Zeichnungen. [18]

Gehäusedesign

Das Design der Rückseiten gestaltet sich einfach, da sich die Geräte nicht unterscheiden und nur Neutriks D-Serien Maße benötigt werden. Diese Maße gibt es bei Neutrik nicht nur als PDF, sondern auch als DXF-Datei, welche für 2D-CAD Anwendungen geeignet sind. Die Signal XLR Anschlüsse habe ich mit 30mm Abstand gesetzt.

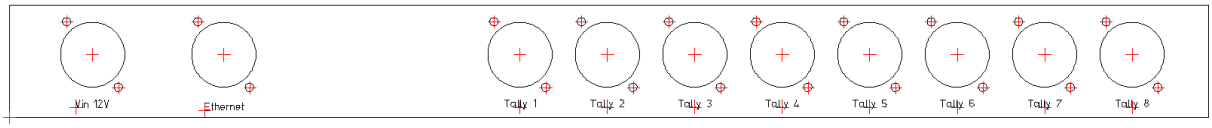

Abbildung 19 – Layout Rückplatte

Sender

Bei den Frontplatten sieht es anders aus. Hier habe ich einen D-Stil Anschluss, eine LED mit 5mm Bohrung und 8 16mm große LED-Taster.

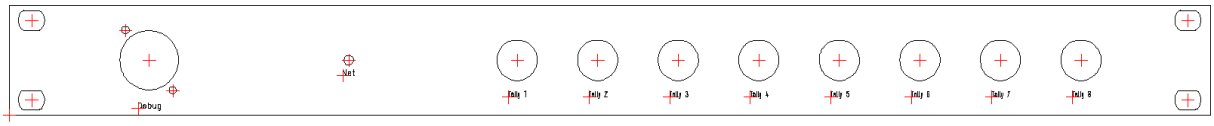

Abbildung 20 – Layout Front Sender

Empfänger

Hier habe ich einen D-Stil Anschluss und 9 LEDs mit einem Abstand von 18mm.

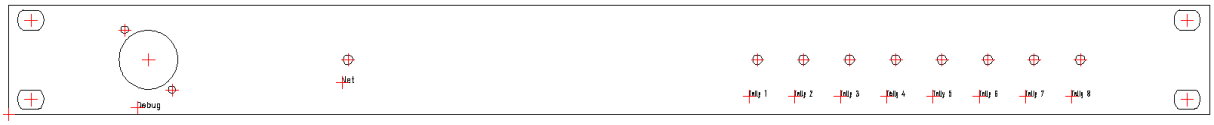

Abbildung 21 – Layout Front Empfänger

Bohren der Löcher

Alle Bohrungen habe ich auf meinen Platten markiert und gebohrt.
Nach Ankunft meiner Rohplatinen, habe ich entsprechende M3 Bohrungen an den Bodenplatten gebohrt.

Abtrennen von Metallteilen

Das Gehäuse hat an der Bodenplatte und am Deckel gebogenes Metall. Es soll dem Gehäuse mehr Stabilität verleihen. Ein Teil dieses Metallstücks ist im Weg. Daher musste ich ca. 50mm Teilstücke mit einem Dremel entfernen, damit der USB-Port Platz hat.

Aufbau der Geräte

Löten der Platinen

Als Erstes habe ich meine Bauteile auf Vollständigkeit geprüft. Danach habe ich mit einem SMD-Lötkolben alle meine SMD-Bauteile verlötet. Die Optokoppler besitzen einen recht großen Abstand zwischen den Füßen. Die SMD Varistoren waren auch einfach anzubringen. Schwieriger hingegen war der Relais-Treiber dieser ist recht klein und besitzt nur geringen Fußabstand. Hier habe ich Kühlpaste unter dem IC verteilt und mit einem großen Lötkolben mit viel Lötwasser verlötet.

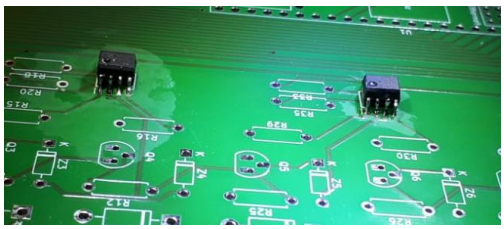

Abbildung 22 – SMD Optokoppler

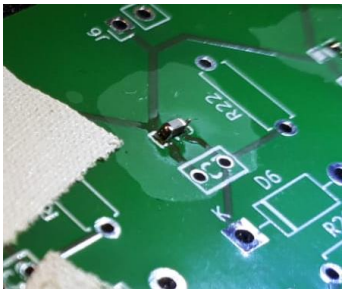

Abbildung 23 – SMD Varistor

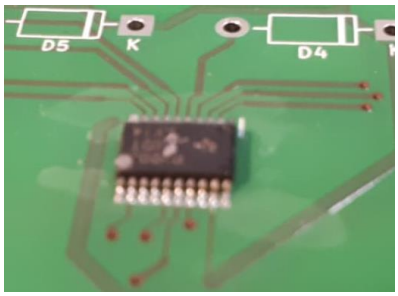

Abbildung 24 – SMD Relais-Treiber

Die THT-Elemente zu verlöten war keine besondere Herausforderung.

Verkabelung

Die Platinen sind mittig im Gehäuse platziert. Die Verkabelung beider Geräte gestaltet sich ähnlich. Die ausgewählten XLR Anschlüsse haben Lötschuhe. In diese sind weibliche Stiftleisten Kabel verlötet und werden auf die Platine gesteckt. Für das Netzwerk wird ein kurzes RJ45 Kabel benutzt.

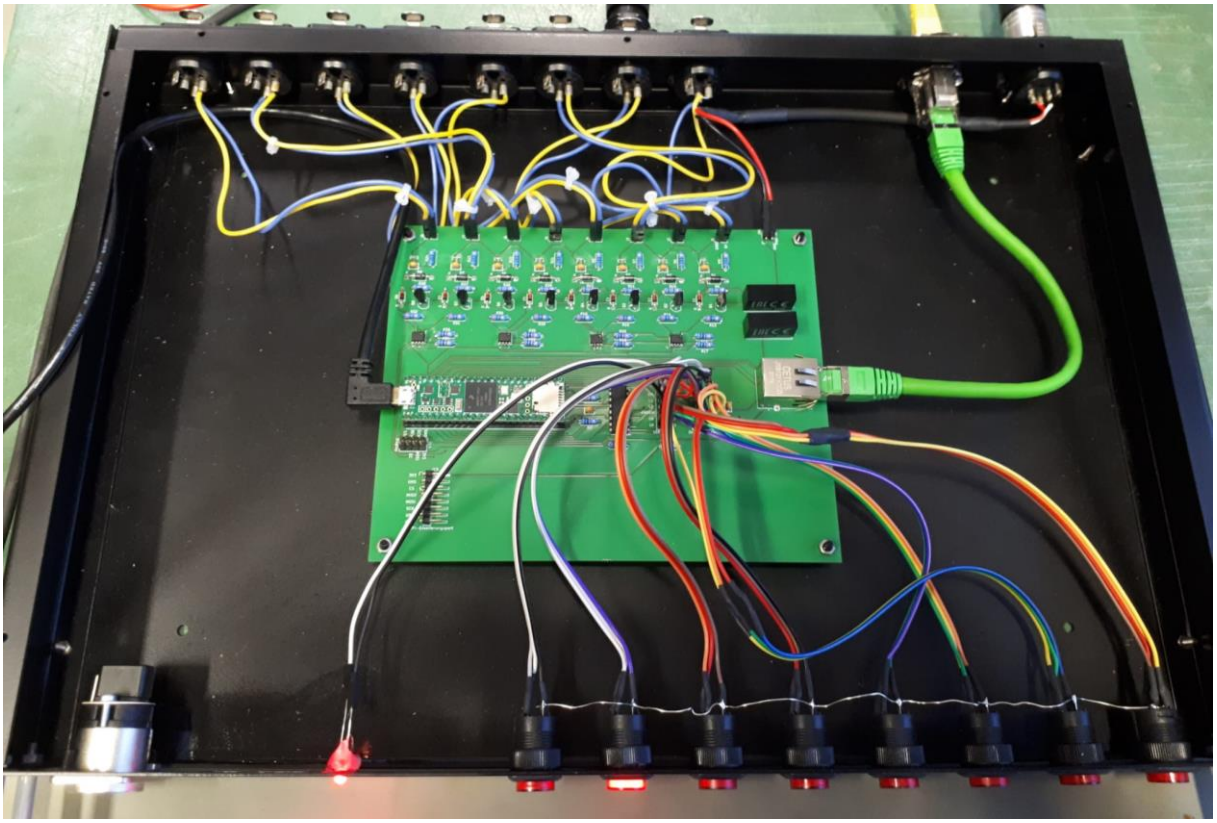

Abbildung 25 – Beispiel Verkabelung Sender

Software

Entwicklungsumgebung

Meine Entwickungsumgebung besteht aus folgenden Elementen:

- Visual Studio Code als IDE/Editor
- *Platformio* als Compiler und Bibliothekenmanager
- *Git* für die Versionierung und Dokumentierung von Änderungen
- *C++* als Programmiersprache für mein Programm

Warum VS Code?

VS Code[19] ist eine schmalere Variante von Visual Studio ohne Compiler. VS Code enthält weiterhin Syntax-Markierung und Überprüfung. Es kann mit unzähligen Plugins erweitert werden. Unter anderem auch Plugins für Git und Platformio[20].

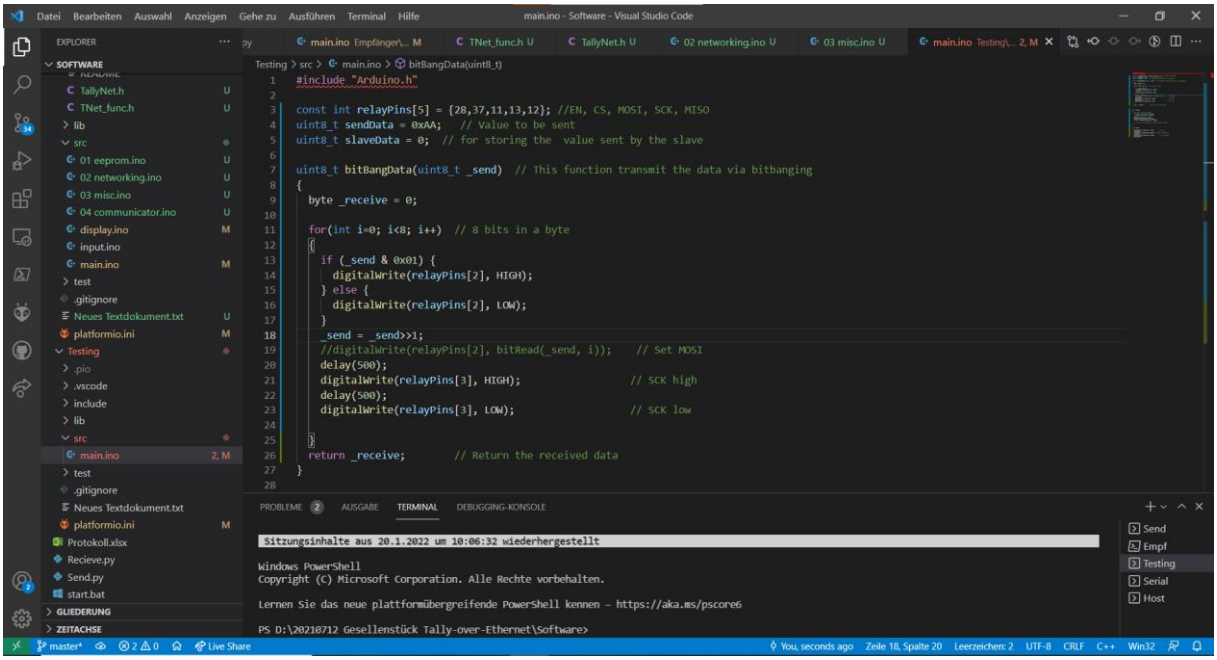

Abbildung 26 – Screenshot von VS Code

Warum Platformio?

Die *Arduino IDE* ist Anfängerfreundlich, allerdings mangelt ihr es an vielen modernen Entwickungsfunktionen. Die Bekannteste ist Multi-File-Support. In ihr kann man Projekte ausschließlich in einer Datei bearbeiten. Statt de Arduino IDE, kann man in der Komination VS Code und Platformio arbeiten.

Doch was ist Platformio?

PlatformIO ist ein plattform- und architekturübergreifendes, professionelles Werkzeug für Ingenieure von eingebetteten Systemen und für Softwareentwickler, die Anwendungen für eingebettete Produkte schreiben.

PlatformIO bietet Entwicklern eine moderne integrierte Entwicklungsumgebung, die plattformübergreifend arbeitet, viele verschiedene Software Development Kits (SDKs) oder Frameworks unterstützt und ausgefeiltes Debugging (Debugging), Unit Testing (Unit Testing), automatisierte Codeanalyse (Static Code Analysis) und Remote Management (Remote Development) umfasst. Es ist so konzipiert, dass die Entwickler maximale Flexibilität und Wahlmöglichkeiten haben, da sie entweder grafische oder Befehlszeilen-Editoren oder beides verwenden können (Beispiel: **`pio run -t upload`** übersetzt den Code und schreibt diesen). [20]
PlatformIO verwendet die Skriptsprache Python. Mit etwas extra Arbeit können auch andere Sprachen als C oder C++ in Platformio verwendet werden.

Warum C++?

Weltweit ist C++ eine der beliebtesten Programmiersprachen für die System- und Anwendungsprogrammierung. C++ ist maschinennah und sehr effizient. Die Sprache eignet sich hervorragend für die Programmierung von Spielen und Desktopanwendungen. Die Effizienz verdankt sie einem umfangreichen Angebot von Sprachelementen und theoretischen Konzepten. Doch der größte Vorteil ist eine breite Community und ein Großangebot von Bibliotheken, die die Entwicklung vereinfacht.

Zentrale Variablen-Vergabe

Ziel ist es, dass alle für den Alltag benötigten Variablen und Einstellungen zentral vergebbar sind und dadurch auch von nicht Programmier-Kollegen abgeändert werden können. PlatformIO besitzt eine Konfigurationsdatei pro Projekt, die platformio.ini. Hier können Endsystem und benötigte Bibliotheken eingetragen werden.

```
FILE: platformio.ini
[env:teensy41]
platform = teensy
board = teensy41
framework = arduino
build_flags = -D USB_SERIAL
lib_deps =
        PaulStoffregen/cores
        vjmuzik/NativeEthernet
        vjmuzik/FNET
        uberi/CommandParser@^1.1.0
```

Zudem habe ich eine Header-Datei in meinem Code, in der viele Programm interne Informationen gespeichert sind.

FILE: TallyNet.h

```
struct TallyNet
{
    int group = 3;                  //Gruppen ID
    const char ident[5] = "TNet";   //Netzwerk Codewort
    uint8_t tally;                  //Tally Status
    int debuglvl = 3;               //Debug Level für Nachrichten
    const int relayPins[5] = {28,37,11,13,12};    //EN, CS, MOSI, SCK,MISO
    const int optoPins[8] = {9,8,7,6,5,4,3,2};              //MSB TallyX
    const int overwPins[8] = {16,17,18,19,20,21,22,23};     //MSB Sx
    const int shiftPins[3] = {29,30,31};  //DATA, RCLK (store), ShiftRCLK
};
```

Netzwerkfunktionalität

Bibliothek NativeEthernet

Die Bibliothek *NativeEthernet* von vjmuzik stellt eine Nutzerfreundliche Schnittstelle zu dem TCP/IP-Stack fNET[21] für den Teensy 4.1 dar. [22]
Ich habe NativeEthernet gewählt, da es sich über die Jahre bewährt hat.

Erweiterung

NativeEthernet ist nur ein Grundgerüst. Alle Verhaltensmuster programmiere ich selbst.

Senden von Unicast Nachrichten

Hierfür habe ich eine Funktion geschrieben:

FILE: Networking.cpp

```
void SendUnicast(IPAddress rip, int rport, char content[], int mode = 1) {
  // if there's data available, read a packet
  Udp.beginPacket(rip, rport);   //uechosrv
  if (mode == 1){
    Udp.write(TN.ident);
    Udp.write(" ");
    Udp.write(TN.group + '0');
    Udp.write(" ");
  }
  Udp.write(content);
  Udp.endPacket();
  delay(10);
}
```

`mode = 1` stell das Codewort und Gruppen-ID vor die Nachricht.

Senden von Broadcast Nachrichten

Diese Funktion ähnelt der Unicast-Funktion. Hier wird nur die IP Addresse mit der Broadcast-Adresse ersetzt.

Empfangen von Nachrichten

FILE: Networking.cpp

```
1 void netReceive()
2 {
3   // Receive and decide handling to subfunctions
4   int packetSize = Udp.parsePacket();
5   if (packetSize)
6   {
7     IPAddress remote = Udp.remoteIP();
8     if (remote == ip)
9     {return;}
10    Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
11    char content[UDP_TX_PACKET_MAX_SIZE];
12    strcpy(content, packetBuffer);
13    memset(&packetBuffer[0], 0, sizeof(packetBuffer)); // Clear buffer
14
15    char *pch;
16    pch = strstr(content, TN.ident); // Finde Schlüsselwort in C-String
17    if (pch != NULL)
18    { // Wenn gefunden, dann:
19      Serial.println(pch);
20      Serial.print("Sizeof Ident: ");
21      Serial.println(sizeof(TN.ident));
22
23      //Isolieren der GruppenID in C-String
24      char idchar[2];
25      idchar[0] = content[sizeof(TN.ident)];
26      idchar[1] = 0;
27      int idnum = atoi(idchar); // Umwandeln des char in int
28      if (idnum == TN.group)
29      { // Wenn GuppenID gleich mit Empfang, dann
30        //Ausgabe des Empfangs
31        Serial.print(remote);
32        Serial.print(":");
33        Serial.println(Udp.remotePort());
34        Serial.print("got command: ");
35        char cmdIn[25];
36        strcpy(cmdIn, content + sizeof(TN.ident) + 2);
37        Serial.println(cmdIn);
38
39        // Analyse des Befehls
40        char response[CmdParser::MAX_RESPONSE_SIZE];
41        netParser.processCommand(cmdIn, response);
42        Serial.println(response);
43      }
44      else
45        return;
46    }
47    else return;
48    return;
49  }
50 }
```

Diese Funktion ist etwas komplizierter. Zuerst prüfe ich, ob ein UDP-Paket angekommen ist. Wenn ja, dann sortiere ich die Pakete mit der eigenen IP als Sender direkt aus. Zeile 10 bis 13 beinhalten das Einlesen der Nachricht. Dann prüfe ich, ob das Codewort und die eingestellte GruppenID übereinstimmen. Zuletzt gebe ich den Befehl für eine Analyse an den Interpreter weiter.

Protokoll

Um die Kommunikation zwischen meinen Geräten zu standardisieren, habe ich ein Protokoll entworfen. Version 1 beinhaltet folgende Funktionen:

Grundlegende Funktion

	Tally-Info			
Art	Richtung			Inhalt
BROADCAST	Sender	-->	Empfänger	[KENNUNG] [GRUPPE] set [TallyBits]
	Request Tally			
Art	Richtung			Inhalt
BROADCAST	Sender	<--	Empfänger	[KENNUNG] [GRUPPE] tallyreq

`,set‘` ist der Befehl zum Übertragen der Tally-Zustände. Er wird nur gesendet, wenn sich ein Tally-Zustand ändert.
`,tallyreq‘` wird vom Empfänger gesendet, wenn dieser gestartet und noch keinerlei Tally-Daten hat. Der Sender antwortet mit einem `,set‘`-Befehl

Erweiterte Kommunikation

	Keep-Alive + Feedback			
Art	Richtung			Inhalt
BROADCAST	Sender	-->	Empfänger	[KENNUNG] [GRUPPE] ping
UNICAST	Sender	<--	Empfänger	[KENNUNG] [GRUPPE] pong

Neben dem ICMP-Ping habe ich zusätzlich diese Art des Pings implementiert.

Management Befehle

	Globale Gruppenänderung			
Art	Richtung			Inhalt
BROADCAST	Sender	-->	Empfänger	[KENNUNG] [GRUPPE] chgrp [NGRUPPE]
UNICAST	Sender	<--	Empfänger	[KENNUNG] [GRUPPE] chgrp ok

Hier kann der Sender oder ein Computer im LAN eine Gruppenänderung ausführen.

Serielle Diagnose

Um die Diagnose und das Einstellen von Werten im laufenden Betrieb zu erleichtern, habe ich mich dazu entschlossen, eine Serielle Konsole zu implementieren. Die Auswahl an Befehlen ist in der aktuellen Version gering und soll in Zukunft erweitert werden.

Implementation der Konsole

FILE: main.cpp

```
1  if (Serial.available() > 0)
2    { // Wenn Serielle Daten anliegen:
3      Serial.print(":  ");
4       // lese Daten bis EOL-Zeichen
5      charsRead = Serial.readBytesUntil('\n', sEingabe, sizeof(sEingabe) - 1);
6      if (sEingabe[charsRead - 1] == '\r') {
7        // Wenn Windows-EOL-Zeichen dann ersetzen mit EOL für c-string
8        sEingabe[charsRead - 1] = '\0';
9      }
10     sEingabe[charsRead] = '\0';
11     Serial.println(sEingabe);
12
13     char response[CmdParser::MAX_RESPONSE_SIZE];
14     SerialParser.processCommand(sEingabe, response);
15     Serial.println(response);
16   }
```

Hier lese ich die Eingabe ein, falls Daten vorliegen. Dann muss ich am Ende meiner char-Array (C-String) ein ‚*End-of-Line*'-Zeichen einfügen. Zuletzt gebe ich mein Befehl an den Komandointerpreter weiter.

Verändern von Werten

Befehl	Argumente	Beschreibung
set	group	Setzt Gruppe
debug	int	Setzt das Debug-Level

Abfrage von Werten

Befehl	Argumente	Beschreibung
?		Listet alle Komandos
help		Listet alle Komandos
read	tally	Gibt den Tallystatus aus
read	settings	gibt TallyNet.h aus

Komandointerpreter

Damit Befehle, die per Netzwerk oder Serieller Konsole kommen, richtig behandelt werden, nutze ich eine Bibliothek für diese Aufgabe.

Bibliothek CommandParser

Funktionen der Bibliothek sind:

- Keine dynamische Speicherzuweisung, dies ist gut für eine lange Laufzeit.
- Stark typisierte Argumente mit strenger Eingabeüberprüfung (einschließlich Erkennung von Integer-Überläufen!).
- Für Nutzer verständliche Fehlermeldungen für ungültige Eingaben (z.B. Parse-Fehler: Ungültiges Double für Arg 2, wenn das zweite Argument des Befehls nicht als Double geparst werden kann).
- Unterstützung für Escape-Sequenzen in String-Argumenten (Regex) (z.B. SOME_COMMAND "\"\x41\r\n\t\").

Diese Bibliothek hat einen höheren RAM-Footprint im Vergleich zu ähnlichen Bibliotheken, da sie jedes Argument vollständig parst, anstatt es als String zu belassen. [23]

Eingangüberprüfung

‚Polling' vs ‚Interrupts'

Für das Einlesen oder Reagieren auf Eingangsveränderungen werden zwei Verfahren verwendet.

Polling: Das Programm überprüft den Zustand regelmäßig und ruft ggf. eine Bearbeitungsfunktion auf.
Interrupt: Gerät „meldet" sich beim Prozessor, der daraufhin in eine Bearbeitungsfunktion verzweigt.
Dabei ist Polling ein zeitgesteuertes System und Interrupts ein ereignisgesteuertes System. Die Vor- und Nachteile von Polling und Interrupts sind folgende:

Polling:
Ereignisbearbeitung erfolgt synchron zum Programmablauf
– Ereigniserkennung über das Programm „verstreut"
– Hochfrequentes Überprüfen; hohe Prozessorlast; hoher Energieverbrauch
+ Implizite Datenkonsistenz durch festen, sequenziellen Programmablauf
+ Programmverhalten gut vorhersagbar

Interrupts:
Ereignisbearbeitung erfolgt asynchron zum Programmablauf

+ Ereignisbearbeitung kann im Programmtext gut separiert werden
+ Prozessor wird nur beansprucht, wenn Ereignis tatsächlich eintritt
– Höhere Komplexität durch Nebenläufigkeit; Synchronisation erforderlich
– Programmverhalten schwer vorhersagbar
– Oftmals geringe Anzahl von Ereignisauslösern

Bei meinen bisherigen Projekten habe ich Interrupts vorgezogen, allerdings ist hier Polling sinnvoll, da ich für das Projekt insgesamt 16 Eingänge auf Veränderungen prüfen muss. Das ist mit Interrupts nicht zu realisieren.

Einlesen der Eingänge

FILE: Misc.cpp

```
1  void UpdateIn() {
2      uint8_t tnum;
3      uint8_t button = 0B00000000;
4      uint8_t opto = 0B00000000;
5      debugMSG("UpdatIN", 3);
6
7      for (int i = 0; i < 8; i++) {        // überprüfe Eingänge
8          if (digitalRead(TN.optoPins[i]) == LOW) {
9              //opto = opto | 0x01;
10             bitWrite(opto, i, 1);
11         } else {
12             //opto = opto & 0xfe;
13             bitWrite(opto, i, 0);
14         }
15         //opto = opto << 1;
16
17         if (digitalRead(TN.overwPins[i]) == HIGH) {
18             //button = button | 0x01;
19             bitWrite(button, i, 1);
20         } else {
21             //button = button & 0xfe;
22             bitWrite(button, i, 0);
23         }
24         //button = button << 1;
25     }
26
27     tnum = opto | button;
28     if (tnum == TN.tally) return;
29     TN.tally = tnum;
30
31     char line[50];
32     int cx;
33
34     cx = snprintf(line, 49, "set %X", tnum);
35
36     debugMSG(line, 3);
37     SendBroadcast(8888, line);
38     send_led(tnum);
39 }
```

Zuerst erstelle ich alle nötigen Variablen. Danach eröffne ich einen **`for`**-Loop. In diesem überprüfe ich alle Eingänge und schreibe das ensprechende Bit mit **`bitWrite`** in mein Byte (**`uint8_t`**). Nach dem ich beide Bytes voll habe, vergleiche ich diese in Zeile 27 mit einem Bit-ODER und speicher die Werte in **`tnum`**. Wenn ich diese Zustände bereits kenne, wird die Funktion beendet. Wenn nicht, wird der neue Wert in **`TallyNet::tally`** abgespeichert und ein mithilfe der **`snprintf`** [24] und meiner **`SendBroadcast`** Funktion passe ich den neuen Status an den Empfänger weiter.

Relaisansteuerung

Die serielle Kommunikation mit dem Relais-Treiber erfolgt im 8-Bit-Format, die Datenübertragung wird mit einem seriellen Takt vom Mikrocontroller synchronisiert (SPI). Ein einziges Register steuert alle Ausgänge (ein Bit pro Ausgang). Standardmäßig sind die Relais aus. Das erste Relais (OUT1) kann über einen seriellen Eingang vom Mikrocontroller oder über den speziellen Freigabe-Pin (EN1) gesteuert werden. Wenn EN1 auf GND gezogen wird oder offen bleibt, steuert der serielle Eingang über das Schieberegister OUT1.

Ansteuerung

FILE: Misc.cpp

```
1 void sendRelay(uint8_t _send)
2 {
3   digitalWrite(relayPins[1], LOW);        // SS low
4   delayMicroseconds(500);
5
6   for(int i=0; i<8; i++)
7   {
8     if (_send & 0x01) {
9       digitalWrite(relayPins[2], HIGH);
10    } else {
11      digitalWrite(relayPins[2], LOW);
12    }
13    _send = _send>>1;
14    delayMicroseconds(50);
15    digitalWrite(relayPins[3], HIGH); // SCK high
16    delayMicroseconds(50);
17    digitalWrite(relayPins[3], LOW);  // SCK low
18  }
19
20  delayMicroseconds(500);
21  digitalWrite(relayPins[1], HIGH);       // SS high again
22  return;
23 }
```

Hier verwende ich die Bit-Bang SPI-Methode, um die Daten an meinen Relais-Treiber zu senden.
Das SS-Signal aktiviert den Empfang von SCLK- und MOSI-Daten, wenn es auf LOW gesetzt wird. Nachdem die 8-Bit-Daten übertragen sind, wird SS wieder auf High gesetzt, um den Empfang von SCLK und MOSI zu deaktivieren und die seriellen Daten an das Steuerregister zu übertragen.
Das Datenblatt gibt zu erkennen, dass SCLK niedrig gehalten werden muss, wenn SS HIGH ist.

LED-Anzeige

Für die Anzeige der Tally Ein- und Ausgaben bei Sender und Empfänger habe ich LEDs beziehungsweise LED-Taster eingebaut. Diese werden mit dem Schieberegister angesprochen.

FILE: Misc.cpp

```
1 void send_led(uint8_t bits) {
2     digitalWrite(TN.shiftPins[1], LOW);
3     // shift out bits:
4     shiftOut(TN.shiftPins[0], TN.shiftPins[2], LSBFIRST, bits);
5     // latch pin auf high damit LEDs angehen:
6     digitalWrite(TN.shiftPins[1], HIGH);
7     delayMicroseconds(500);
8     digitalWrite(TN.shiftPins[1], LOW);
9     delayMicroseconds(500);
10 }
```

Der Befehl ‚**shiftOut**‘ schiebt ein Byte von Daten hinaus. In diesem Fall beginnt dieser mit dem äußersten rechten Bit.

Bugfixing

Fehlerbehebung des Relais-Treibers

Problem

Beim Design der Platine habe ich nicht aufgepasst und den Relais-Treiber falsch angeschlossen.

In Abbildung 28 aus dem Datenblatt wird sugeriert, dass der Not-RST Pin ein Ausgang am Treiber ist. Dies stimmt allerdings nicht.

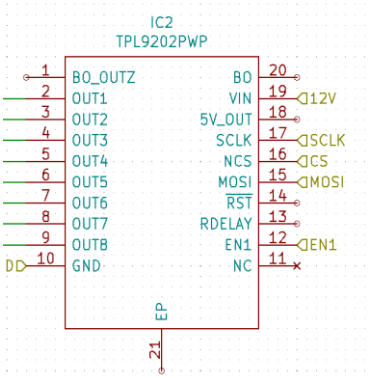

Abbildung 27 – Ausschnitt Relais-Treiber in Schaltplan

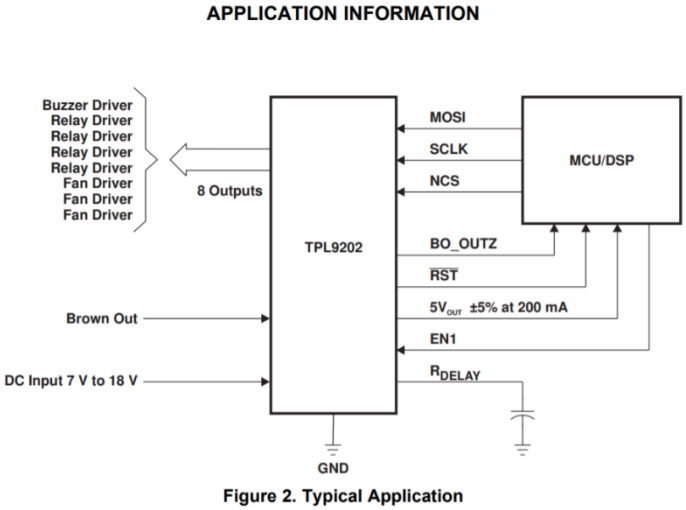

Abbildung 28 – Ausschnitt Relais-Treiber Datenblatt

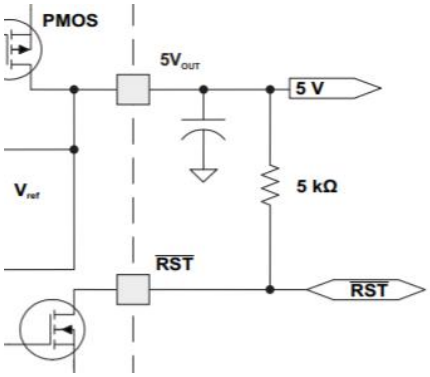

Abbildung 29 – Ausschnitt Funktionsdiagramm

Denn ein genauerer Blick auf das Funktionsdiagramm im Datenblatt zeigt einen 5kΩ Widerstand zwischen 5Vout und Not-RST (siehe Abbildung 29).

Lösung

Damit ich den IC dennoch ansteuern kann, musste ich einige Verbindungen und Bauelemente an die Pins des ICs löten. Auf der Rechten ist die neue Schaltung zu finden. Zudem musste ich GND auf der Sekundärseite beider Spannungsisolatoren miteinander verbinden.

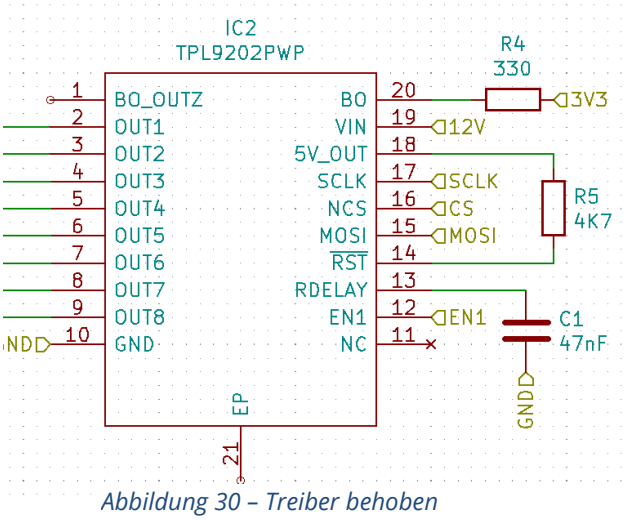

Abbildung 30 – Treiber behoben

Reihenfolge der Funktionsdeklaration

Es kann vorkommen, dass eine Funktion eine andere Funktion aufruft, bevor sie vom Compiler hinzugefügt wurde. Wenn viele Funktionen in verschiedenen **cpp**-Dateien geschrieben werden, muss oftmals die Reihenfolge für die Deklaration festgelegt werden. Somit werden Abhängigkeiten erfüllt, ohne den Inhalt der Funktion anzugeben und der Compiler ist bedient. Der Inhalt der Funktionen füllt der Compiler beim Einlesen der Quellcodedateien automatisch.

Meine Lösung war die Erstellung einer Header-Datei mit allen zu deklarierenden Funktionen und die folgende Einbindung mit **#include <TNet_func.h>**
In die **main.cpp** Datei.

FILE: TNet_func.h

```
1 #include <Arduino.h>
2 void StartNet();
3 void SendBroadcast(int rport, char content[], int mode = 1);
4 void SendUnicast(IPAddress rip, int rport, char content[], int mode = 1);
5 void netReceive();
6 void netPing();
7 void netRelpyPong();
8 void netCountPong();
9
10 void send_led();
11 void UpdateIn();
12 void initPins();
13 void initnetParser();
14 void initserialParser();
15 void UpdateOut();
16 void send_relay(uint8_t data);
17 void debugMSG(char content[], int level);
```

Temporäre Fehlerbehebung Taster

Wird ein Taster in der aktuellen Platinen-Version gedrückt zieht dieser den Eingang der Teensy auf HIGH (active-high). Wird dieser wieder losgelassen, hält sich die Ladung auf dem Teensy-Eingang hoch genug, um weiterhin als HIGH zu gelten. Das liegt an den fehlenden Pull-Down Widerständen.

Temporär habe ich die Überprüfung der Taster-Eingänge auf eine analoge Abfrage mit hohem Schwellenwert behoben.

Aussicht

Verbesserungsvorschläge

Mir sind einige Punkte bei der Herstellung meines Gesellenstücks aufgefallen, die verbessert oder hinzugefügt werden können.

Überarbeitung der Hardware

Diese Punkte werden in der nächsten Version behoben.

Filter nach Spannungswandler

Die eingesetzten Spannungsisolatoren geben eine sehr saubere und zuverlässige Spannung aus, selbst wenn sich die Eingangsspannung um ±2V abweicht. Dennoch kann ein zusätzlicher LC-Filter nach den Isolatoren eingebracht werden.

Klemmterminals statt Jumper für Spannung

Eine weitere Überlegung ist die 90Grad Stiftleisten für den Spannungseingang der Platine durch Klemmterminale zu ersetzen.

Sicherung vor und nach Spannungswandler

Ein Überstromschutz vor und nach den Spannungsisolatoren ist sinnvoll. Im Fall einer Fehlfunktion, wird dadurch nur die Sicherung betroffen sein. Es ist viel schwieriger, die Spannungsisolatoren, ICs oder Mikrocontroller auf einer Außenübertragung zu ersetzen, als eine Sicherung.

LEDs Vorwiderstand versetzen

Aktuell existiert nur ein Vorwiderstand für alle LEDs / Taster. Das hat zur Auswirkung, dass LEDs dunkler oder heller werden, je nach dem wie viele LEDs aktiviert sind. Lösung ist ein Vorwiderstand pro LED, statt einen für alle LEDs. Zudem sollten die Widerstände größer dimensioniert werden, um die Leuchtkraft zu reduzieren und die Lebensdauer der LEDs zu erhöhen.

Fehlerhafte Beschriftungen abändern

Während meines Aufbaus des Gerätes sind mir mehrere Fehlbeschriftungen aufgefallen, diese sollten beseitigt werden. Bei der Beschriftung von der Sender-Platine wurde GND und +12V vertauscht.

[Sender] Taster von PullUp auf PullDown ändern

Derzeit wird ein Taster mit einer analogen Port-Lesung über eine Active-High Logik angesteuert. Die Implementierung ist fehlerhaft. Ich werde die Taster auf eine Active-Low Logik (Pull-Down, negative-Logik) umändern.

[Empfänger] Diverse Änderungen an Relaistreiber

Am Relais-Treiber wird die Lösung (siehe Abbildung 30) direkt auf der Platine korrigiert. Zudem werden die GND Anschlüsse der Isolatoren zusammengeführt.

Alternativ dazu sollte über eine andere Methode der Ansteuerung der Relais nachgedacht werden, um die Trennung der zwei Spannungsisolatoren wieder zu gewährleisten. Eine Ansteuerung des ICs mit Optokopplern oder die direkte Steuerung der Relais ohne den Treiber-IC wäre denkbar.

Kühlungs-DuKos einfügen

Unter dem Relais-Treiber-IC sind mehrere Durchkontaktierungen (DuKos) und auf der gegenüberliegenden Seite ist eine Kupferfläche für Kühlung des ICs anzusiedeln. Zudem soll der IC mit einem Lötofen angebracht werden, anstatt der hier genutzten Hand-Verlötung.

Größere Leiterbahnen

Die zu dem IC führenden Leiterbahnen sollten auch in der Breite erweitert werden, sodass ggf. ein höherer Strom ohne Temperaturerhöhung fließen kann.

EN1 an GND anschließen

Um sicher zu gehen, dass der Zustand an EN1 eindeutig und nicht schwebend/offen ist, soll dieser direkt an GND angeschlossen werden.

Softwareänderungen

Diese Punkte werden in der nächsten Version behoben.

Erweiterung der Managementeigenschaften

Befehle der seriellen Konsole und Netzwerkkonsole sollen erweitert werden. So sollen in der seriellen Konsole Befehle zum Einstellen von DHCP oder anderen Werten im laufenden Betrieb hinzugefügt werden.

Im Netzwerk sollen neben der jetzt bereits möglichen Gruppenänderung auch die Ausgabe von Einstellungen, weiteren Infos und setzen von einigen IP-Einstellungen möglich werden. Alternativ kann über eine Implementierung eines durch einfachen Webinterface nachgedacht werden.

Umstellung auf normales C++

Derzeit basiert ein großer Anteil des Codes auf Arduino-spezifischen C++ Befehlen. Diese sollen durch eigene Funktionen nach und nach ausgetauscht werden. Ziel ist es ausschließlich ein autonormal ISO- oder GNU-gerechtes C / C++ im Code und Bibliotheken zu verwenden. Dies soll unter anderem die Lesbarkeit und Weiterentwicklung fördern.

Zudem wird mir dieses Vorgehen in Zukunft von Vorteil sein, da ich mit ISO/GNU C++ mehr Erfahrung habe. Allerdings muss ich mich weiter in die Low-Level Funktionen der Teensy-/Arduino-/ESP-Mikrocontrollern einlesen.

Umstellung auf QNEthernet

QNEthernet von ssilverman[25] basiert, anders als NativeEthernet (fNET IPstack), auf dem lwIP TCP/IP-Stack. Dieser ist um einiges leichter als fNET, könnte dadurch weitere Latenzen reduzieren. lwIP unterstützt dafür einige Protokolle nicht. So fehlt die Unterstützung für IPv6, welche in Zukunft wichtig werden könnte.

Implementieren von Netzwerk-Protokollen

Unabhängig von einem IP-Stack Wechsel sollen Technologien wie AutoIP, http-Server oder Telnet eingefügt werden.

Der Port von fNET auf den Teensy ist noch unvollständig und befindet sich noch in Entwicklung. Sollte in Zukunft der ganz fNET-Stack zur Verfügung stehen, könnte dieser beispielsweise die Interpreter-Bibliothek ersetzen. Die Folge wäre weniger Programmcode und schnellere Laufzeiten.

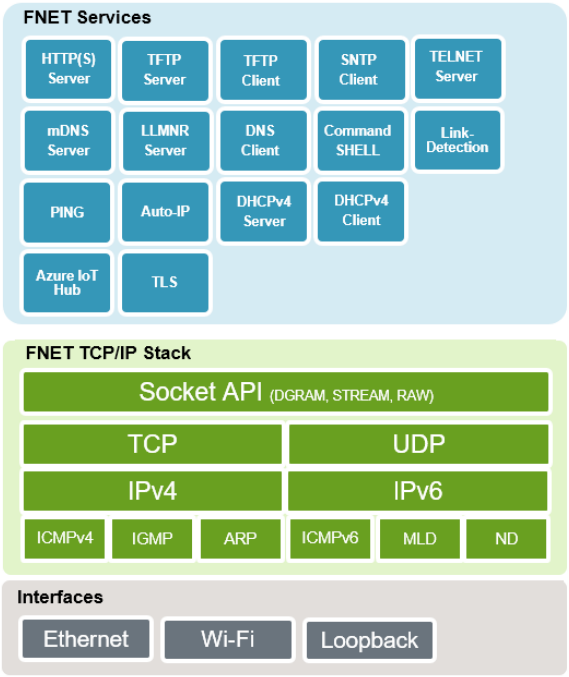

Abbildung 31 – Auszug vollständiger fNET-Stack

Erweiterungsmöglichkeiten

Funkmodule (LoRa)

In den Empfänger könnte ein LoRa-Funkmodul eingebaut werden, um kleine tragbare Tally-Empfänger für einen Ausgang zu realisieren.

Display für Schnelldiagnose

Ein Display für die Anzeige von Statusmeldungen ist sinnvoll, somit muss nicht immer ein Computer angeschlossen werden, um zu wissen, ob keine IP bezogen werden kann.

Tragbare Tally-Empfänger

Wie oben erwähnt, könnten mit LoRa gesteuerte Tally Empfänger realisiert werden.

Eine weitere Idee: Kleine Tally Empfänger ohne Funk, mit einen Ausgang, Ethernet-Anschluss und ggf. Power-over-Ethernet.

Anhänge

Dieser Arbeit liegen alle Schaltpläne, Platinenlayouts und Gehäusedesigns im DIN A3 Format ohne Seitenangaben bei. Auf Anfrage können KiCAD-, LibreCAD-Dateien und Programmcode digital bereitgestellt werden.

Schlusswort

Nach ca. 4,5 Monaten an dem Projekt „Mehrfach-Tally Übertragung über Ethernet“ oder kurz „TallyNet“ ist es nun so weit: Meine Ausbildung nähert sich dem Ende. Alle versprochenen Punkte, die ich im Projektantrag genannt habe, wurden erfüllt.
Obwohl ich einige schöne Wintertage und Wochenenden an meinem Computer oder in meinem zweiten Arbeitsplatz, meinem Bastelkeller verbrachte, fand ich bis auf wenige Momente die Arbeit und das direkte Testen am Projekt sehr spannend und ich bin froh dieses Projekt gewählt zu haben.

Bedanken möchte ich mich bei dem Prüfungsausschuss-Vorsitzenden Herrn Dieter Bork für die Unterstützung bei technischen Fragen, sowie bei Herrn Tobias Weis für die Versorgung mit Spezialwerkzeug und technischem Know-how, wenn es mal nicht so funktioniert hat wie es sollte. Zudem möchte ich mich bei meinen Ausbildern Herrn Magnus Schmitt und Herrn Jörg Hofmann für die jahrelange Unterstützung während der Ausbildung bedanken. Danke an Thomas Zaunbrecher und Alexander Schumann für das Einbringen der Idee, die schließlich zu diesem Projekt geworden ist.

Herzlichen Dank auch an alle Personen & Fachbereiche, die an meiner Ausbildung mitgewirkt und mich über die Zeit der Ausbildung unterstützt haben.

Quellverzeichnis

1 Vgl. Mücher, Michael: Online-Lexikon - BET, in: BET, 01.04.2021, https://www.bet.de/lexikon/rotlicht (abgerufen am 18.06.2021).

2 Vgl. Zettl, Herbert: Television Production Handbook, 12th, 12. Aufl., 2014, https://www.google.de/books/edition/Television_Production_Handbook_12th/yOvKAgAAQBAJ, S. 96.

3 Vgl. Wurtzel, Alan/John Rosenbaum: Television Production, 4. Aufl., 1995, https://www.google.de/books/edition/_/mbJkAAAAMAAJ, S. 407.

S. 3 Vgl. Wikipedia-Autoren: Tally, in: Wikipedia, 01.04.2007, https://de.wikipedia.org/wiki/Tally (abgerufen am 18.06.2021).

4 Vgl. Stoffregen, Paul: Teensy 4.1, in: PJRC, 01.03.2021, https://www.pjrc.com/teensy/ (abgerufen am 18.12.2021). https://www.pjrc.com/teensy/techspecs.html (abgerufen am 18.12.2021).

5 Vgl. FriendlyARM: NanoPi NEO CORE, in: wiki.friendlyarm.com, 01.12.2017, https://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO_Core (abgerufen am 18.12.2021).

6 Texas Instruments: SN74HC595N, in: ti.com, 10.2021, https://www.ti.com/lit/ds/symlink/sn74hc595.pdf (abgerufen am 18.12.2021).

7 Broadcom: HCL-073L, in: broadcom.com, 29.03.2016, https://www.broadcom.com/products/optocouplers/industrial-plastic/digital-optocouplers/100kbd/hcpl-073l (abgerufen am 18.12.2021).

8 Texas Instruments: TPL9202, in: ti.com, 10.2021, https://www.ti.com/lit/ds/symlink/tpl9202.pdf (abgerufen am 18.12.2021).

9 Elektronik-Kompendium: Snubber: http://www.dse-faq.elektronik-kompendium.de/dse-faq.htm#F.25.1 (abgerufen am 19.12.2021).

10 Mean Well: SPUN-05, in: meanwell-web.com, https://www.meanwell-web.com/content/files/pdfs/productPdfs/MW/SPUN02/SPUN02,DPUN02-spec.pdf (abgerufen am 19.12.2021).

11 Mean Well: MDS20, in: mouser.de, https://www.mouser.de/datasheet/2/260/MDS20_spec-1772011.pdf (abgerufen am 19.12.2021).

12 Neutrik: Diverse Anschlüsse, https://www.neutrik.com (abgerufen am 20.12.2021).

13 Wikipedia: XLR, 15.01.2022, https://de.wikipedia.org/wiki/XLR (abgerufen am 21.01.2022)

14 KiCAD: https://www.kicad.org/ (abgerufen am 21.01.2022)
15 Github freerouting: https://github.com/freerouting/freerouting (abgerufen am 21.01.2022)

16 Vgl. Seifert, Tim: General purpose tally interfacing, https://www.cameratim.com/electronics/tallies/ (abgerufen am 21.01.2022)

17 https://aisler.net/

18 https://librecad.org/ (abgerufen am 11.2021)

19 Wikipedia: VS Code, 01.11.2021, https://de.wikipedia.org/wiki/Visual_Studio_Code (abgerufen am 21.01.2022)

20 PlatformIO: https://docs.platformio.org/en/latest/what-is-platformio.html (abgerufen am 21.01.2022)

21 fNET: https://fnet.sourceforge.io/ (abgerufen am 21.01.2022)
22 Github NativeEthernet: https://github.com/vjmuzik/NativeEthernet (abgerufen am 21.01.2022)

23 Github CommandPaser: https://github.com/Uberi/Arduino-CommandParser (abgerufen am 21.01.2022)

24 C++ Reference: https://www.cplusplus.com/reference/cstdio/snprintf/ (abgerufen am 21.01.2022)

25 Github QNEthernet: https://github.com/ssilverman/QNEthernet (abgerufen am 21.01.2022)

Abbildungsverzeichnis