OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Ostbayerische Technische Hochschule Regensburg
Fakultät für Elektro- und Informationstechnik

Projektarbeit/VMCM

Konzeption und Hardwareaufbau eines Dual-CAN (FD) Demonstrators mit Failure Injection Board und TFT-Touchdisplay

Vorgelegt von: Tobias Langer
Matrikelnummer:
Studiengang: Master Elektro- und Informationstechnik

Betreuer: Prof. Dr.-Ing. Hans Meier
Abgabedatum: 29.06.2021

Abstract

Controller Area Network (CAN) ist in automatisierten Bereichen der Industrie ein wichtiger Bestandteil der Kommunikation vieler einzelner Komponenten. Spezielle Anwendungen fordern einen erhöhten Standard hinsichtlich Sicherheit und Fehlertoleranz. Eine Steigerung der Zuverlässigkeit des Systems ist somit unumgänglich.
Die vorliegende Arbeit befasst sich mit der Untersuchung eines Dual-CAN-Bussystems. Hierzu zählt die Entwicklung und Umsetzung eines Demonstratoraufbaus. Es werden fünf Nucleo STM32H7 Boards um ein sogenanntes Failure Injection Board (FIB) und ein 3,5" TFT-Touchdisplay erweitert sowie redundant miteinander über zwei speparate CAN verbunden.
Mithilfe des FIBs wird das duale CAN-Netzwerk realisiert. Außerdem können die Abschlusswiderstände per Knopfdruck zugeschaltet und Drahtbrüche durch Relais auf dem redundanten CAN-Bus simuliert werden. Als weitere Anzeige- und Bedienmöglichkeit wird ein Touchdisplay auf das Nucleo STM32H7 Board gesteckt. Hier soll eine GUI eine intuitive Bedienung ermöglichen und den aktuellen Status der weiteren Teilnehmer im Netz anzeigen.
Das FIB dient zur Entwicklung von Algorithmen um Fehler bzw. Drahtbrüche zu umgehen oder auch andere Busstrukturen zu untersuchen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADC	Analog Digital Converter
CAN	Controller Area Network
CPLD	Complex Programmable Logic Devices
CPU	Central Processing Unit
ESD	Electrostatic Discharge
FIB	Failure Injection Board
FID	Failure Injection Demonstrator
FIM	Failure Injection Module
FPGA	Field Programmable Gate Array
GND	Ground
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
ID	Identifier
IDE	Integrated Development Environment
LCD	Liquid Crystal Display
RM	Redundanzmanager
SPI	Serial Peripheral Interface
TFT	Thin-Film Transistor
TTL	Transistor-Transistor Logik
VMCM	Vertiefung Microcontroller Master

1. Einleitung

In dieser Arbeit wird der Aufbau und die Funktion eines Demonstrators zur Kommunikation mittels eines redundanten CAN-Busses beschrieben. Der sogenannte Dual-CAN basiert auf zwei eigenständigen CAN-Kommunikationskanälen, die parallel betrieben werden. Bei Ausfall eines Kanals soll so weiterhin eine stabile Kommunikation gewährleistet und ein Ausfall des Gesamtsystems verhindert werden. Mithilfe des Demonstratoraufbaus (siehe Abb. 1.0.1) und des FIBs soll so die Kommunikation getestet und verbessert werden. Das FIB kann gezielt Drahtbrüche im Dual-CAN simulieren. Diese Injektionsstelle wird mit LEDs sowie auf dem Display angezeigt.

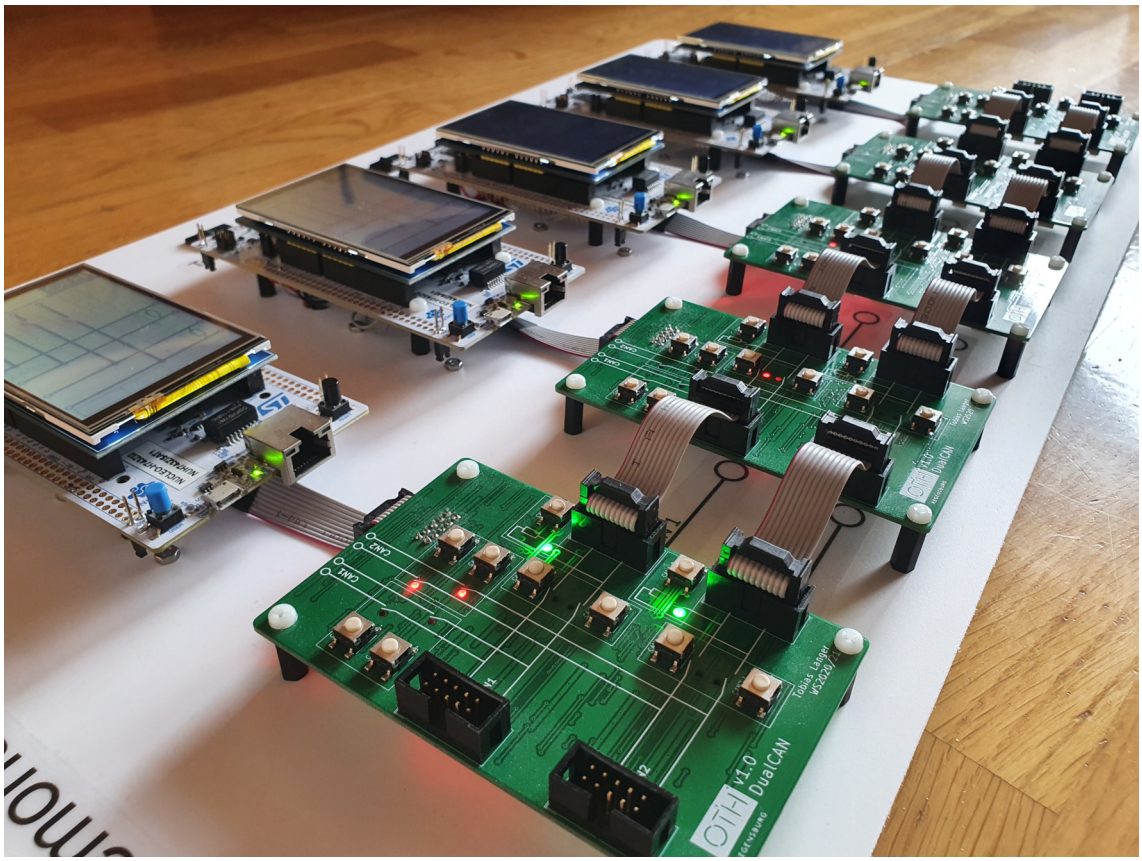


Abbildung 1.0.1: Demonstratoraufbau Dual-CAN

1.1 Definition Dual-CAN

CAN ist ein zuverlässiger und kostengünstiger Feldbus für Echtzeitanwendungen mit einer bereits sehr niedrigen Ausfallwahrscheinlichkeit (siehe Kapitel 2.2.1). Aufgrund der differentiellen Zweidrahtleitung, der zyklischen Redundanzprüfung (CRC Check) und dem Bit-Stuffing wird gewährleistet, dass die Nachrichten korrekt übertragen werden und der Bus nahezu fehlerfrei funktioniert. Bei den meisten Anwendungen besteht, aufgrund der geringen Ausfallwahrscheinlichkeit, kein weiterer Handlungsbedarf. Bei sicherheitskritischen Situationen, wie z.B. in der Raum- oder Luftfahrttechnik, besteht dennoch Handlungsbedarf. [1]
Um die Zuverlässigkeit des Systems zu erweitern wird ein redundantes CAN-Bussystem eingeführt (Dual-CAN), z.B. für zukünftiges autonomes Fahren. Abb. 1.1.1 zeigt das Blockschaltbild des aufgebauten Netzwerks.

Dual-CAN bietet durch die doppelte Ausführung der Kommunikationskanäle eine erhöhte Betriebssicherheit und damit eine geringere Ausfallwahrscheinlichkeit.
Es ist somit sinnvoll Dual-CAN in den genannten Anwendungsgebieten zu verwenden, die sicherheitskritische Aufgaben übernehmen, schwierig zu warten sind oder auf korrekte Datenübertragung angewiesen sind.

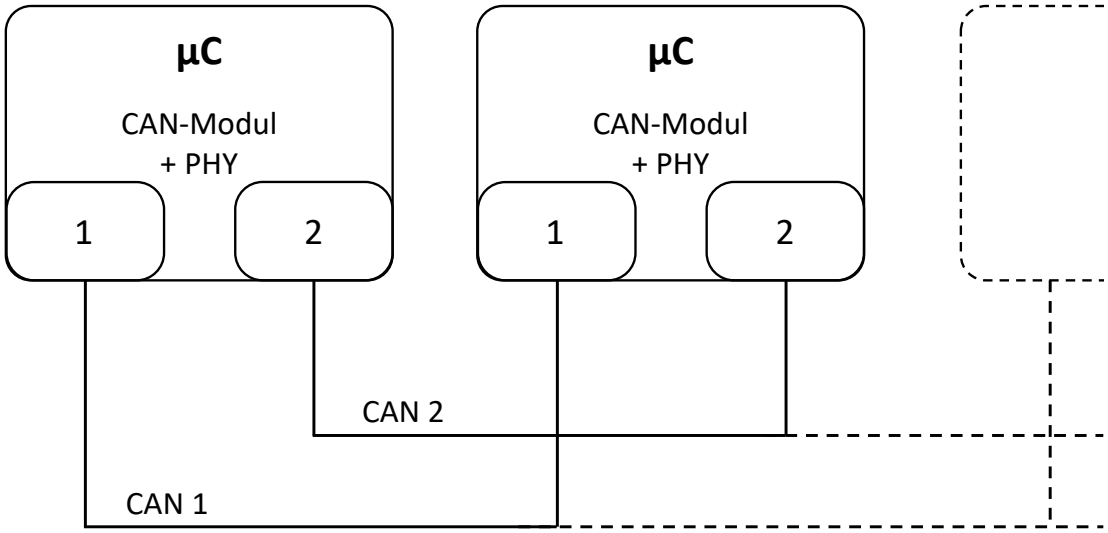


Abbildung 1.1.1: Aufbau mit zwei eigenständigen CAN-Bussen CAN_1 und CAN_2

1.2 Problemstellung und Zielsetzung

Ausgangslage dieser Projektarbeit sind fünf Microcontroller von STMicroelectronics (Nucleo STM32H7), die mittels CAN kommunizieren sollen. Dieses CAN-Netzwerk soll redundant ausgeführt sein und zwei Kommunikationskanäle besitzen. Um das redundante Bussystem auf das Verhalten bei Drahtbrüchen untersuchen zu können, muss es eine Möglichkeit geben, gezielt Drahtbrüche zu simulieren. Es soll dazu eine Platine entwickelt werden, die genau diese Aufgaben erledigen kann.
Das entstehende FIB übernimmt nun die Fehlersimulation mittels Relais und zeigt die Drahtbrüche durch LEDs an. Außerdem werden auf der Platine zwei CAN-Transceiver angebracht, um die Kommunikation zu ermöglichen.

Im weiteren Verlauf soll zudem ein Touchdisplay mit je einem Microcontroller verbunden werden. So soll die Visualisierung und Bedienung des Demonstratoraufbaus (Failure Injection Demonstrator (FID)) intuitiver und einfacher gestaltet werden.
In Abb. 1.2.1 ist der Aufbau eines Moduls dargestellt. Das sogenannte Failure Injection Module (FIM) besteht aus Microcontroller, Thin-Film Transistor (TFT)-Touchdisplay und FIB.

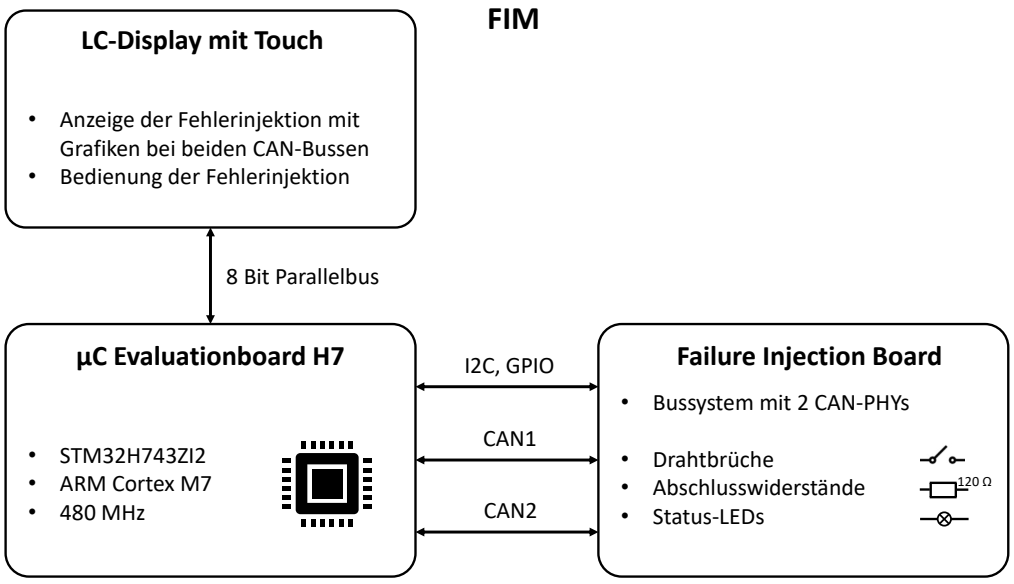


Abbildung 1.2.1: Komponente FIM des Demonstrators

1.3 Aufbau der Arbeit

Diese Arbeit beschäftigt sich zu Beginn mit den Grundlagen von CAN, dessen Funktionsweise und Anwendungsgebiete, sowie die Erweiterung zum redundanten Dual-CAN in Kapitel 2. Anschließend wird in Kapitel 3 das FIB genauer beschrieben. Im Vordergrund stehen hier die verwendeten Bauteile, sowie das Platinendesign und die Ansteuerung der Platine. Die Anzeige und Bedienung mittels TFT-Touchdisplay des Dual-CAN Demonstators wird in Kapitel 4 erläutert. Das Gesamtsystem und die Zusammenarbeit der einzelnen Bauteile wird in Kapitel 5 nähergebracht. Abschließend wird in Kapitel 6 und in Kapitel 7 die Validierung und Fazit sowie ein Ausblick zu Verbesserungen und möglichen Erweiterungen wiedergegeben.
Die Projektarbeit erstreckt sich vom Aufbau des Demonstrators über das Platinendesign des FIB zur Programmierung des Dual-CAN-Bussystems. Das Thema für Vertiefung Microcontroller Master (VMCM) beinhaltet die Ansteuerung des Displays und Ausgabe von Bilddateien sowie die Steuerung des FIB über das Touchdisplay.

2. Grundlagen

Auf den folgenden Seiten werden allgemeine Grundlagen zu CAN, dessen Funktionsweise und Anwendungsgebiete sowie die Erweiterung zu Dual-CAN erläutert. Weiterhin wird ein Einblick in die Redundanz von Bussystemen gegeben und dazu die Ausfallwahrscheinlichkeiten im Vergleich zu einem einfach ausgeführten Bussystem untersucht und dargestellt.

2.1 Controller Area Network (CAN)

Das serielle Bussystem CAN übernimmt in Fahrzeugen und Industrieanlagen wichtige Aufgaben zur Datenübermittlung. Es wurde entwickelt, um die Datenübertragung zu vereinfachen, zu beschleunigen und sicherer zu gestalten. Seit 1994 ist CAN standardisiert und erlaubt Datenraten von bis zu 1 MBit/s bei einer Netzwerkausdehnung von maximal 40 Metern. Die Erweiterung zu CAN FD (Flexible Data-Rate) ermöglicht eine Übertragungsgeschwindigkeit der Daten von bis zu 8 MBit/s. [2, 3]

2.1.1 Funktionsweise

Dem ISO/OSI-Modell entsprechend arbeitet CAN nur auf dem Physical Layer und dem Data Link Layer und ist somit sehr einfach und schlicht aufgebaut.
In Abb. 2.1.1 ist der prinzipielle Aufbau eines CAN-Netzwerks dargestellt und zeigt zudem auch die Notwendigkeit der beidseitigen Abschlussterminierung mit Widerständen von je 120 Ω.
Das CAN-Netzwerk beschreibt eine Linientopologie und ist eine Multi-Master-Architektur. Jeder Knoten ist im Prinzip gleichberechtigt und darf zu jeder Zeit seine Botschaften auf den Bus schicken. Die gesendeten Nachrichten sind mit einem sog. Identifier (ID) gekennzeichnet und beschreibt die Priorisierung der Nachrichten (je niedriger die ID, desto höher die Priorität der Nachricht). Die ID kann im Standard oder Extended Format mit 11 oder 29 Bits gesendet werden. Aufgrund des CSMA/CA Prinzips (Collision Avoidance) wird so ein niedrig priorisierter Sendeversuch abgebrochen und erneut gestartet, wenn der Bus wieder frei ist. [2]

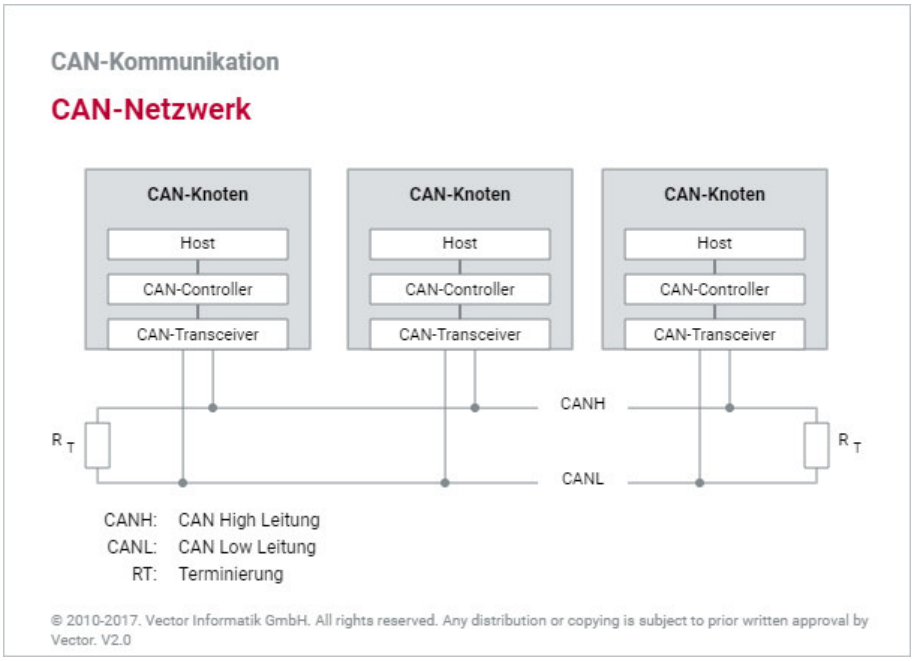


Abbildung 2.1.1: Prinzipieller Aufbau eines CAN-Netzwerks [2]

Ein Knoten besteht aus einem CAN-Controller und einem CAN-Transceiver. Der Controller realisiert das vorgeschriebene Übertragungsprotokoll und deren Kommunikationsfunktionen um, während der Transceiver die Spannungspegel auf die typischen CAN_Low und CAN_High Signale umsetzt (siehe Abb. 2.1.2). Die Signalübertragung funktioniert auf einer sogenannten Differenzsignalübertragung mit Twisted-Pair-Leitungen. Es können so elektromagnetische Störungen unschädlich gemacht werden.

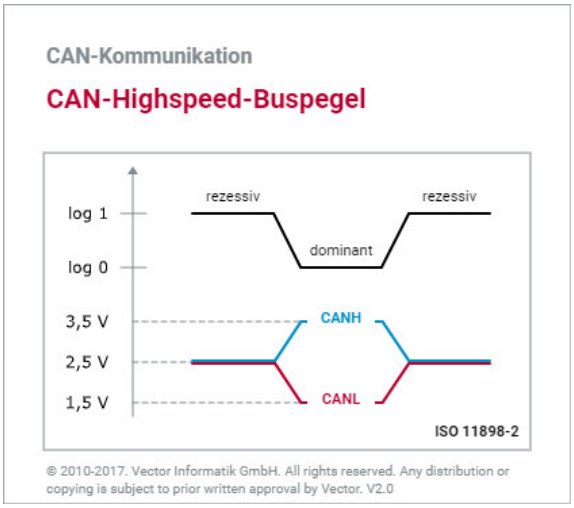


Abbildung 2.1.2: Typische Spannungspegel in einem CAN-Netzwerk [2]

Weiterhin zählen zu den wichtigen Merkmalen des CAN-Busses die zyklische Redundanzprüfung (CRC Check) und das Bit-Stuffing. Es kann so die Integrität der Übertragung der Daten sichergestellt werden. Die Rendundanzprüfung stellt die korrekte Datenübertragung sicher, während beim Bit-Stuffing nach jedem 5. Bit der gleichen Polarität ein entgegengesetztes eingefügt wird, um die Abtastpunkte synchron zu halten.
Um die Datenkonsistenz darüber hinaus sicherzustellen, darf jeder Teilnehmer eine Nachricht als fehlerhaft kennzeichnen und melden. Hierzu werden die Sende- und Empfangsfehlerzähler eingesetzt, die bei korrekt empfangenen bzw. gesendeten dekrementiert und nach gewissen Regeln bei fehlerhaften Nachrichten erhöht werden. In Abhängigkeit der Zählerstände besitzen die Teilnehmer unterschiedliche Zustände, die Error Active, Error Passive und Bus Off benannt werden. Beim Zustand Bus Off wird der Teilnehmer komplett vom Netz genommen. [1, 2]

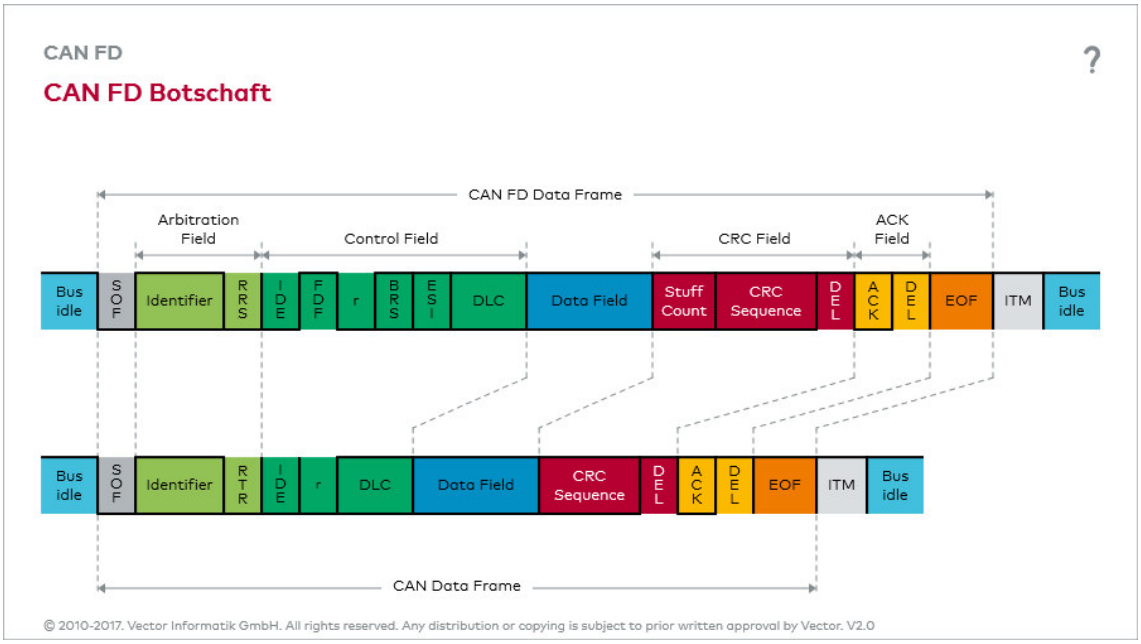


Abbildung 2.1.3: Aufbau einer CAN-Nachricht (unten) bzw. CAN FD-Nachricht (oben) [2]

In Abb. 2.1.3 ist unten der normale CAN Data-Frame dargestellt. Dieser besteht im Grunde aus der ID, dem Data Length Code (DLC), den Daten selbst und der CRC Sequenz.
Erweitert wurde der CAN Standard durch die Flexible Data-Rate (FD) wodurch auch eine höhere Buslast bewältigt werden kann. CAN FD-Controller sind abwärtskompatibel und senden nur das Datenfeld mit einer höheren Geschwindigkeit. Es ergeben sich nur minimale Änderungen im Data-Frame, diese können ebenfalls der Abb. 2.1.3 entnommen werden. [2]
Der Vorteil von CAN FD macht sich bei einer hohen Datenmenge durch das soge-

nannte Bit-Rate-Switching deutlich. Es wird bspw. der Header mit einer Geschwindigkeit von 500 kBit/s und die Daten mit 2 MBit/s gesendet. [4]

2.2 Dual-CAN und Redundanz

Aufgrund der vorher erwähnten Mechanismen des CAN besitzt der Bus bereits eine sehr hohe Verlässlichkeit. Da jedoch die Hardwarebauteile eine geringe, aber dennoch vorhandene Ausfallwahrscheinlichkeit aufweisen, besteht bei sicherheitskritischen Anwendungen weiterhin Handlungsbedarf. Beispiele hierzu sind die Raum- oder Luftfahrt. Hier ist meist die Wartung sehr aufwendig oder gar nicht möglich und es muss zu jeder Zeit ein Ausfall der Komponenten verhindert werden. [5, 6].
In dieser Arbeit wird ein zweites, parallel betriebenes und baugleiches CAN-Netzwerk hinzugefügt (siehe Abb. 3.0.1). Dies erhöht die Fehlertoleranz des ganzen Systems und macht es möglich Drahtbrüche auf dem Bus zu erkennen, zu melden oder sogar zu umgehen. [7]

2.2.1 Arten der Redundanz

Um die Grundlagen des Dual-CAN beschreiben zu können, werden in diesem Kapitel verschiedene Techniken von redundanten Bussystemen beschrieben. Diese Techniken beschreiben die Kommunikationsstrecke zwischen zwei Teilnehmern (d.h. Mikrocontroller, CAN-Controller, CAN-Transceiver, Busleitung, CAN-Transceiver, (Redundanzmanager), CAN-Controller, Mikrocontroller). (siehe Abb. 2.2.3)
Es gibt unterschiedliche, teils sehr aufwändige Stufen, um eine Redundanz zu realisieren. Die einfachste Methode ist die, nur die Treiberbausteine und das Übertragungssystem redundant (doppelt oder mehrfach) zu realisieren. Weiter wäre es möglich auch die CAN-Controller mehrfach auszuführen. Als aufwändigste Stufe, kann man die Central Processing Unit (CPU) bzw. auch die Spannungsversorgung redundant realisieren, um die Ausfallwahrscheinlichkeit zu minimieren. [8]
Im Nachfolgenden werden die zwei am häufigsten verwendeten Redundanzrealisierungen genauer erläutert und eine Formel zur Berechnung der jeweiligen Verlässlichkeit $R(t)$, welche die Qualität der fehlerfreien Funktion des Bussystems beschreibt, eingeführt. λ ist hier die Ausfallwahrscheinlichkeit des Übertragungssystems, λ_s ist die des Busselektors (Redundanzmanagers) (mehr hierzu in Kapitel 2.2.2).

Cold Redundancy

Bei der sogenannten Cold Redundancy verweilen die redundant verbauten Bauteile im Standby-Betrieb und werden aktiviert, sobald im Hauptkommunikationskreis ein Fehler erkannt wird. Oft werden hier die Fehlerzähler der einzelnen CAN-Teilnehmer verwendet. Werden Nachrichten nicht mehr korrekt empfangen oder gesendete Nachrichten nicht mehr bestätigt, ist dies ein Zeichen für eine fehlerhafte Kommunikation

(Fehlerzähler). Es wird der Kommunikationskanal gewechselt und der redundante Kanal benutzt. Hierbei kann es zu Übertragungsverlusten kommen, da der Wechsel zwischen den beiden Kanälen nicht sofort bzw. nur mit einer bestimmten Verzögerung erfolgen kann. Die Vorteile hingegen sind der niedrige Energieverbrauch und auch die vergleichsweise einfache bzw. vom CAN-Protokoll bereits vorhandene Implementierung. Zudem wird hier keine zusätzliche Hardware, wie bei der in Kapitel 2.2.2 erwähnten Methode benötigt. In der Formel (2.1) wurde zum Vergleich dennoch der Redundanzmanager berücksichtigt. [6, 9]
Verlässlichkeit $R(t)$ [10]:

$$R(t) = e^{-\lambda_s t}(e^{-\lambda t} + \lambda t e^{-\lambda t}) \tag{2.1}$$

Hot Redundancy

Im Gegensatz zur Cold Redundancy werden die Bauteile des Hauptkommunikationskanals und die des redundanten Kanals immer parallel betrieben. Es wird jede Nachricht, die vom Mikrocontroller gesendet wird, parallel auf beiden Bussystemen ausgegeben. Der Empfangsmechanismus gestaltet sich hier deutlich schwieriger, als bei der Cold Redundancy. Es muss geprüft werden, welcher Kanal richtig funktioniert und somit auch welche Nachricht zum Mikrocontroller weitergeleitet werden soll. Der Energieverbrauch ist bei dieser Methode höher, was aber durch die geringere Ausfallwahrscheinlichkeit und höhere Verlässlichkeit des Systems, aufgrund der stets korrekten Übermittlung der Nachrichten, kompensiert werden kann. [6, 9]
Verlässlichkeit $R(t)$ mit n Anzahl der redundanten Systeme (bei Dual-CAN: $n = 2$) [10]:

$$R(t) = e^{-\lambda_s t}(1 - (1 - e^{-\lambda t})^n) \tag{2.2}$$

Anwendung bei Dual-CAN

Die meist verwendete Realisierung der Redundanz ist die Hot Redundancy Methode. Es kann hier gewährleistet werden, dass in sicherheitskritischen Systemen keine wichtigen Informationen verloren gehen und auch im teilweise fehlerbehafteten Betrieb die Kommunikation weitergeführt wird. Gleichzeitig könnte das System gewartet und der Fehler wieder behoben werden. Die gewählte Methode der Redundanz sorgt für eine erhebliche Erhöhung der Verlässlichkeit und der Stabilität der Nachrichtenübertragung.

In dieser Arbeit wird der Dual-CAN ebenfalls als Hot Redundant ausgeführt, jedoch ohne einen Redundanzmanager, wie in Kapitel 2.2.2 beschrieben. Dieses Kapitel dient als allgemeine Darstellung der Funktion eines redundanten Bussystems.

2.2.2 Funktionsweise

In diesem Kapitel wird die Hot Redundancy Strategie und dessen Realisierung genauer betrachtet. Der Fokus liegt hierbei auf der am häufigsten eingesetzten Hardware, sowie die Realisierung der Nachrichtenumleitung und Fehlererkennung in der Software.

Aufbau

Die grundsätzlich benötigte Hardware ist ähnlich zum einfachen CAN. Benötigt wird ein Mikrocontroller für die Steueraufgaben und zur Interpretation der Nachrichten, ebenso wie der CAN-Controller und -Transceiver auf der Kommunikationsebene. Der CAN-Controller implementiert das CAN-Protokoll, wobei der CAN-Transceiver die Signale auf die typischen CAN_Low und CAN_High Pegel umsetzt. Das im Folgenden beschriebene Redundanzprinzip hat eine doppelte Ausführung des CAN-Transceivers. Der CAN-Controller und der Mikrocontroller sind nur einfach verbaut.

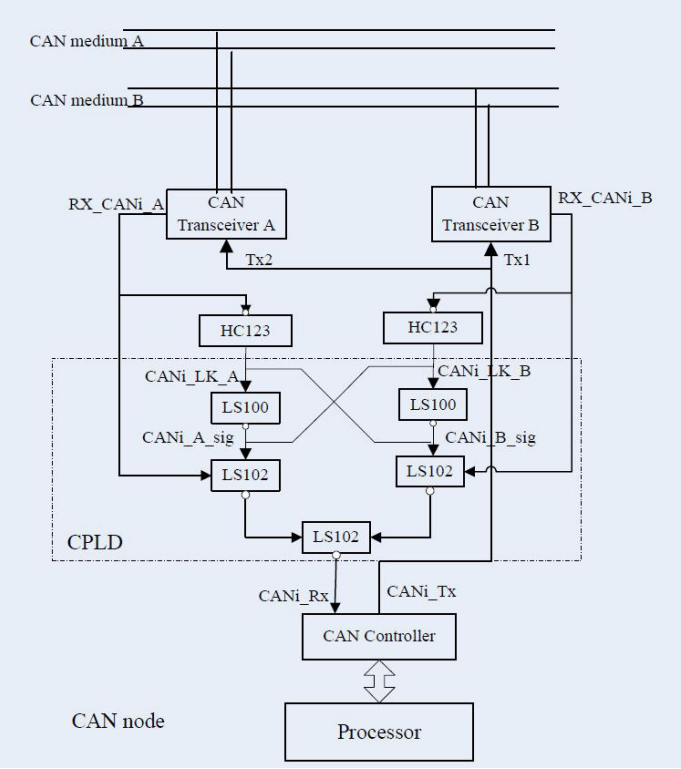


Abbildung 2.2.1: Aufbau eines redundanten CAN-Netzwerks [11]

Wie in Abb. 2.2.1 erkennbar ist, wird zusätzlich ein sog. Redundanzcontroller benötigt. Dieser wird als programmierbare Hardware ausgeführt (Complex Programmable Logic Devices (CPLD) oder Field Programmable Gate Array (FPGA)). Es können Prozesse parallel abgearbeitet und so der Programmablauf zum Empfang von Nachrichten realisiert werden. Aufgrund von möglichen Verzögerungen errei-

chen die Nachrichten auf den beiden Bussystemen nie gleichzeitig die beiden CAN-Transceiver. Daraus folgt, dass die empfangenen Nachrichten speziell ausgewertet und verglichen werden müssen.
Das Senden der Nachrichten erfolgt direkt (ohne Redundanzcontroller). Die Tx-Signale des CAN-Controllers werden ohne Umwege auf die Transceiver weitergegeben und gesendet. [11]

Empfang einer Nachricht:

Der in Abb. 2.2.1 beschriebene Aufbau prüft, in welchem Kanal die Nachricht als erstes ankommt und setzt mithilfe des HC123 Bausteins eine logische 1 für die Dauer des Empfangs der Nachricht. Dieses Bauteil verlängert also den ursprünglichen Startimpuls der empfangenen Nachricht. Wenn aufgrund von Fehlern auf dem Bus ein Kanal gestört ist, wird dieser aufgrund der Verzögerung oder durch Ausbleiben der Nachricht als der langsame Kanal gewertet und die Nachrichten werden über den ungestörten Kanal verarbeitet. [11]

Nachrichtenumleitung und Kanalwechsel:

In Abb. 2.2.2 wird der Mechanismus zur Nachrichtenumleitung deutlich. Im markierten Teil 2 und 4 kann man erkennen, dass genau die Nachricht an CAN_RX gesendet wird, dessen Kanal als erstes eine korrekte Nachricht empfängt. Teil 3 verdeutlicht das Prinzip, dass der Empfang einer Nachricht erst abgeschlossen werden muss, bevor es dem anderen Kanal erlaubt wird, seine Nachrichten zu empfangen und an den Mikrocontroller weiterzuleiten (hier Kanal RX_CAN1_B gesperrt). Außerdem ist die durch den Redundanzmanager (RM) verursachte Verzögerung an der Verschiebung zwischen den Signalen RX_CAN1_A und CAN_RX in Teil 2 zu erkennen. Die Nachrichtenumleitung funktioniert also automatisch durch den Redundanzcontroller, wie in Abbildung 2.2.3b dargestellt. [11]

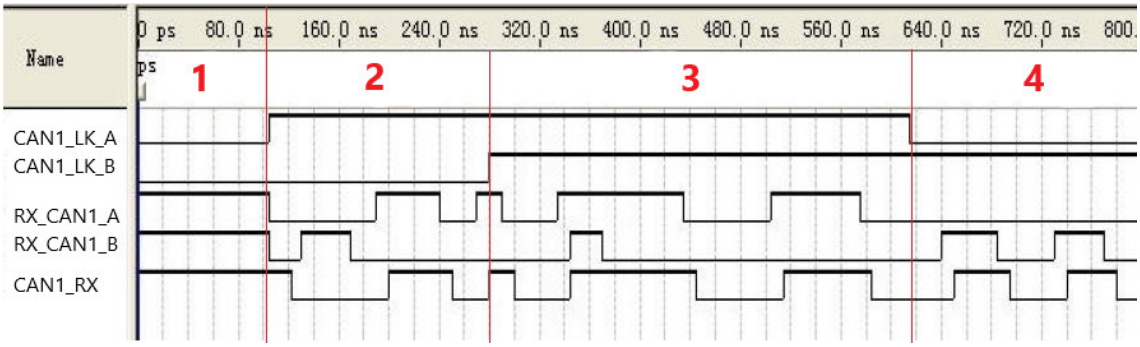


Abbildung 2.2.2: Timingdiagramm der empfangenen Nachrichten und Prinzip der Nachrichtenumleitung auf den anderen Kanal [11]

2.2.3 Berechnung der Ausfallwahrscheinlichkeiten

Im Folgenden wird der Dual-CAN in den Punkten Sicherheit und Verlässlichkeit genauer betrachtet und ausgewertet.

Verlässlichkeit / funktionale Sicherheit

Nachfolgend ist der Aufbau der Netzwerke schematisch dargestellt und die Verlässlichkeit den einzelnen Bauteilen zugewiesen. Berechnet wird die Verlässlichkeit des normalen CAN-Bus und des Hot Redundant Modus mit den Formeln (2.3) (zugehörig zu Abb. 2.2.3a) und (2.4) (Abb. 2.2.3b). Abb. 2.2.3c zeigt den in dieser Arbeit verwendeten Aufbau des Dual-CANs. Es werden zwei getrennt voneinander betriebene Buskanäle gezeigt, ohne Redundanzmanager. Somit ergibt sich die Formel für diese Ausfallwahrscheinlichkeit nach Formel (2.4) ohne λ_6. [6]

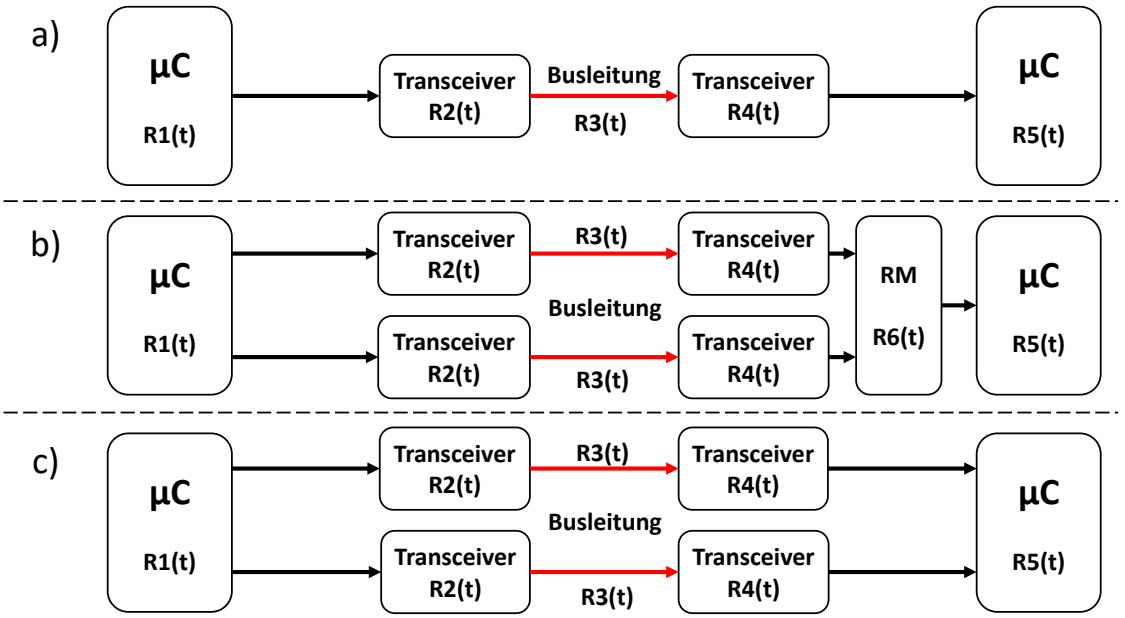


Abbildung 2.2.3: Verlässlichkeit R(t) der einzelnen Bauteile - a) Single-CAN, b) Dual-CAN mit Hardware-Redundanzmanager, c) Dual-CAN ohne Hardware- Redundanzmananger (basiert auf Software) (angelehnt an [6] und ergänzt um c))

Die in Abb. 2.2.3 dargestellten Verlässlichkeiten werden in den Formeln bereits summiert. Die Ausfallwahrscheinlichkeiten λ_n gehören zu den jeweiligen Verlässlichkeiten R_n.
λ_s aus Kapitel 2.2.1 entspricht hier λ_6. Die Formel (2.2) der Hot Redundancy wurde hier um die entsprechenden Bauteile erweitert und formt nun Formel (2.4).

$$R(t)_{single} = e^{-(\lambda_1+\lambda_2+\lambda_3+\lambda_4+\lambda_5)t} \tag{2.3}$$

$$R(t)_{dual} = e^{-\lambda_1 t}[1 - (1 - e^{-(\lambda_2+\lambda_3+\lambda_4)t})^2]e^{-(\lambda_5+\lambda_6)t} \tag{2.4}$$

Als Ausfallwahrscheinlichkeiten werden im Paper [6] die Werte von $\lambda_n = 0,00001/h$ verwendet. Da die Wahrscheinlichkeit höher ist, dass ein Fehler in der Übertragungsleitung auftritt, wird $\lambda_3 = 0,0001/h$ gesetzt. In Abb. 2.2.4 ist die berechnete Verlässlichkeit über die logarithmisch aufgetragene Zeit dargestellt. Wie zu erkennen ist, wirken sich die Vorzüge des Dual-CAN mit steigender Betriebsdauer immer stärker aus. So ist schon bei 100 Betriebsstunden ein deutlicher Vorteil zu erkennen.

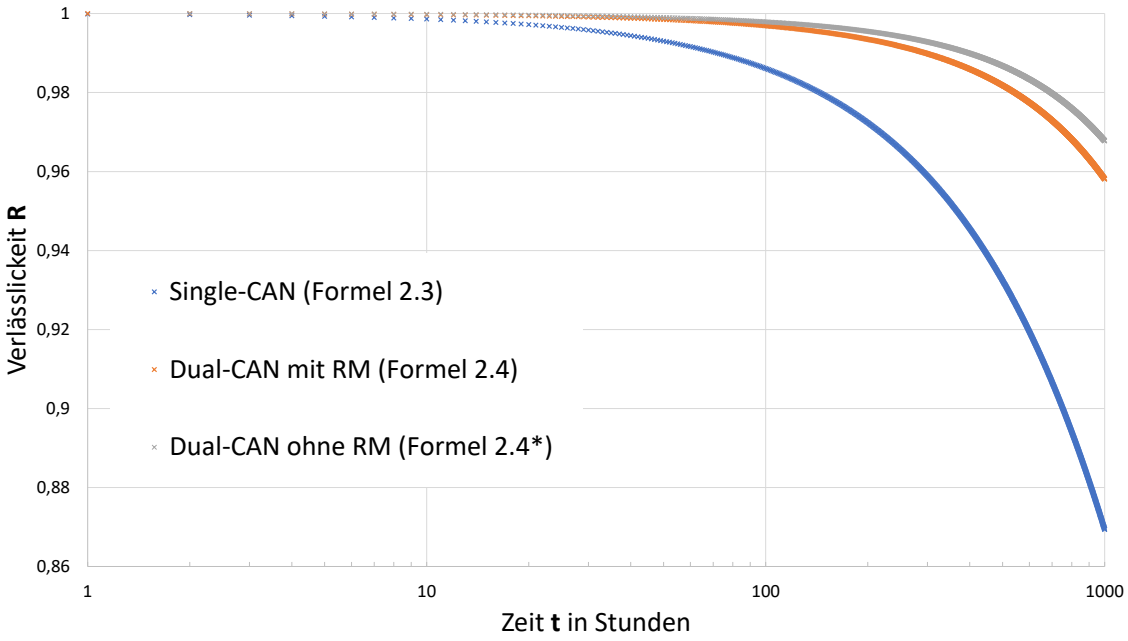


Abbildung 2.2.4: Verlässlichkeit R(t) als Funktion über die logarithmisch aufgetragene Zeit t – Vergleich Single-CAN und Dual-CAN (angelehnt an [6] und ergänzt)
**: Formel ohne λ_6*

2.2.4 Anwendungsgebiete

Der Dual-CAN kommt dort zum Einsatz, wo erschwerte Bedingungen und erhöhte Anforderungen an Sicherheit und Zuverlässigkeit einen fehlersicheren Bus verlangen. Im Folgenden werden drei beispielhafte Einsatzgebiete für Dual-CAN aufgezeigt.

Raumfahrt - Weltraum-Roboterarm

Im Paper [5] kommt der Dual-CAN zur fehlertoleranten Signalübertragung zwischen den Gelenken eines chinesischen Roboterarms im Weltraum zum Einsatz. Aufgrund der Umgebungsbedingungen, wie z.B. Protonen und Ionen, die im Weltall den Roboterarm und dessen Elektronik ungeschützt treffen, muss hier ein redundantes System eingesetzt werden. Da die Reparatur des Roboterarms durch die Bedingungen im Weltall erschwert wird, entsteht so durch Dual-CAN eine zusätzliche Absicherung der Kommunikation im Fehlerfall.
Die in Kapitel 2.2.2 beschriebenen Funktionen des Dual-CANs wird ebenfalls bei obigem Weltraum-Robotern eingesetzt. Es wurde speziell hierfür ein Test- und Kommunikationssystem entwickelt. [11]

Luftfahrt - Avioniksystem von kleinen Flugzeugen

Um in kleinen Flugzeugen die Verlässlichkeit der Kommunikation der Bauteile und der Anzeigen im Cockpit zu verbessern, wurde in [6] ebenfalls ein Dual-CAN entwickelt und getestet. Zwei verschiedene Ansätze werden hier diskutiert und ausgewertet. Die erste Methode wurde mit redundanten CAN-Controllern realisiert, die andere Methode wurde mit einem Redundanzcontroller wie im Beispiel von Kapitel 2.2.2 aufgebaut (ähnlich zum FPGA).

Seefahrt

Dual-CAN eignet sich ebenfalls zur Datenübertragung auf Schiffen. Hierzu wird ein redundantes CAN-Netzwerk eingesetzt, um von der Schiffsbrücke den Hauptantrieb mit allen nötigen Geräten zu steuern. Auch hier sollte ein Ausfall der Kommunikation aufgrund der eingeschränkten Möglichkeiten zur Reparatur unbedingt vermieden werden. [12]

Autonomes Fahren

Um sicheres autonomes Fahren zu ermöglichen, wäre der Dual-CAN eine möglicher Kommunikationskanal. Die Daten können auf dem redundant ausgeführten Bussystem mit einer ausreichend hohen Verlässlichkeit gesendet werden. Die Idee eines autonom-agierenden unbemannten Unterwasserfahrzeugs in [13] ist durchaus auf autonome Fahrzeuge im Straßenverkehr übertragbar.

3. Failure Injection Board (FIB)

In diesem Kapitel wird näher auf die benötigte Hardware eingegangen, um ein redundantes Dual-CAN Netzwerk aufzubauen. Mithilfe des entwickelten FIBs kann beim Dual-CAN gezielt an verschiedenen Stellen ein Drahtbruch simuliert und so die Reaktion des Bussystems beobachtet und analysiert werden.

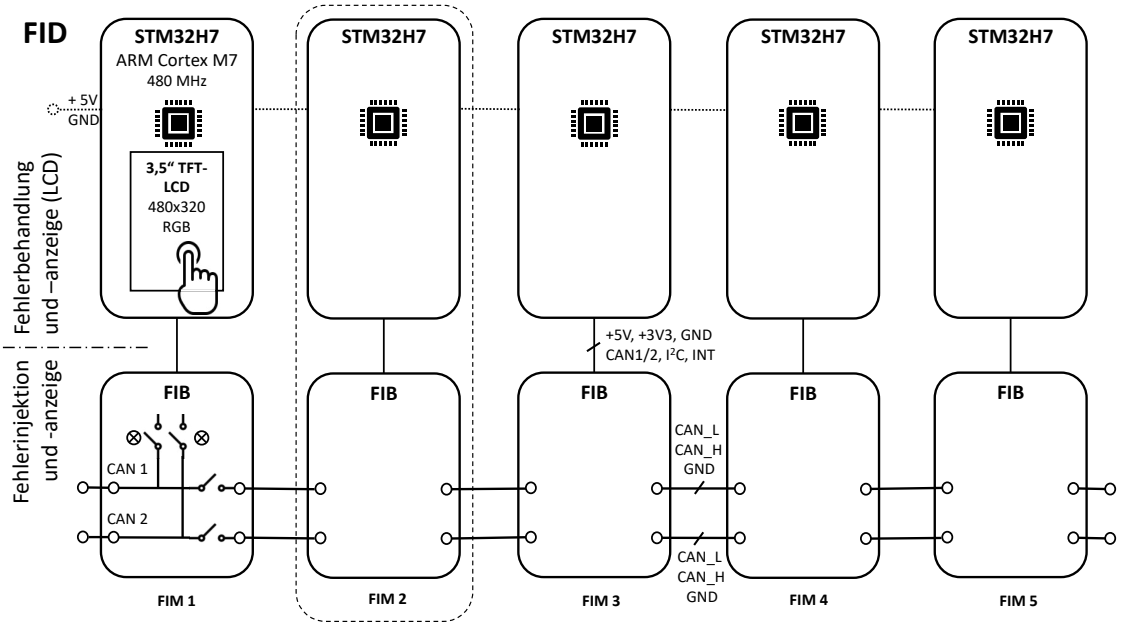


Abbildung 3.0.1: Blockschaltbild - Aufbau des Demonstratoraufbaus (FID) mit Evalutationboard H7, 3,5"TFT.LCD als Aufsteckmodul sowie dem FIB

Der schematisch dargestellte Aufbau des Dual-CAN-Netzwerks, der aus beliebig vielen FIM bestehen kann, ist in Abb. 3.0.1 zu sehen. Im Netz sind fünf Teilnehmer, die mittels CAN untereinander kommunizieren können. Als Steuereinheit wird ein Evaluationboard mit einem Microcontroller von ST verwendet.

Die CAN-Transceiver (PHYs) und die Fehlerinjektionseinheiten sitzen auf den FIBs, die wiederum über ein 10-poliges Flachbandkabel mit dem Microcontroller verbunden sind. Es werden die beiden CAN-Signale für CAN1 und CAN2 auf Transistor-Transistor Logik (TTL)-Ebene sowie General Purpose Input/Output (GPIO)-Interrupt, Inter-Integrated Circuit (I^2C)-Bus und Spannungsversorgung (+5 V und +3,3 V) übertragen.
Die Fehlerinjektionen sind manuell per Taster oder Touch möglich sowie automatisch per Programm für einen Dauerversuch. Durch die Schalter über dem FIM lässt sich das ganze Modul spannungsfrei schalten.
Verbunden sind die FIBs untereinander durch den Dual-CAN-Bus und besitzen eine zentrale Spannungsversorgung. Dies ist ebenfalls pro Bus mit einem 10-poligen Flachbandkabel realisiert. Hier werden nur die entsprechenden Adern für CAN_Low und CAN_High sowie Ground (GND) verwendet. An den äußeren Enden des Demonstrators sind Möglichkeiten zur Erweiterung des Netzwerks oder auch zur Messung und Analyse des CAN-Bus gegeben.
In Abb. 3.0.2 ist der fertige Demonstratoraufbau abgebildet.

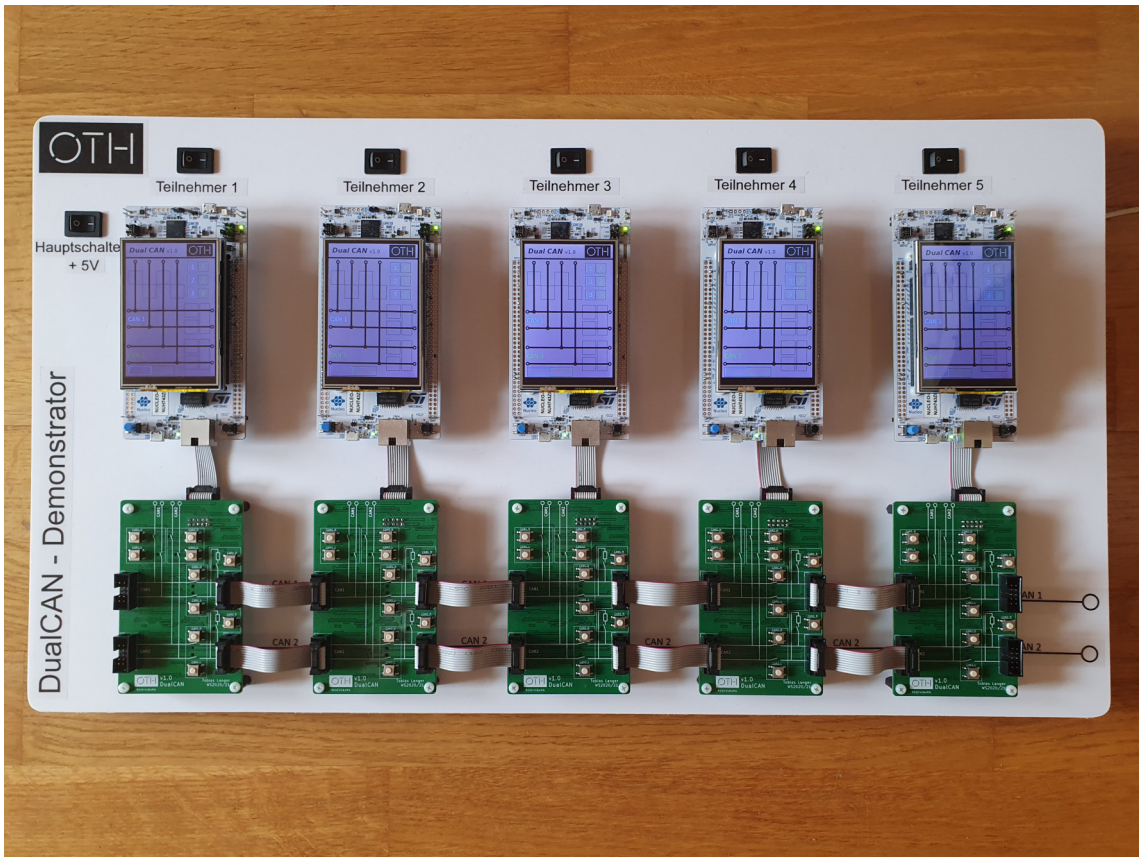


Abbildung 3.0.2: Gesamtaufbau des FID: Evaluationboards H7 mit aufgestecktem LC-Display, FIB und CAN-Verbindungen

3.1 Microcontroller STM32H7

Als Teilnehmer im Netzwerk werden Microcontroller von ST (ARM Cortex M7) eingesetzt. Genauer wird das Evaluierungsboard NUCLEO-STM32H743ZI2 [14] verwendet, welches in Abb. 3.1.1 dargestellt ist.
Die Aufgaben des Microcontrollers bestehen grundsätzlich aus den folgenden drei Punkten:

- über den redundant aufgebauten Dual-CAN zu kommunizieren
- die Steuerung des FIBs (Relais und LEDs) zu übernehmen
- den Status der auf dem FIB verbauten Relais und LEDs zu anzeigen
- Bedienung über Touchdisplay ermöglichen

Außerdem wird auf das Evaluierungsboard mit dem Nucleo-144 Pinout ein Display, welches näher in Kapitel 4 betrachtet wird, zur besseren Bedienung aufgesetzt. Die verwendbaren GPIO-Pins sind in Abb. 3.1.2 bis 3.1.5 dargestellt und einsehbar. Die verwendeten Pins wurden farblich gekennzeichnet und hervorgehoben (lila: externe Spannungsversorgung, rot: Display, grün: FIB).

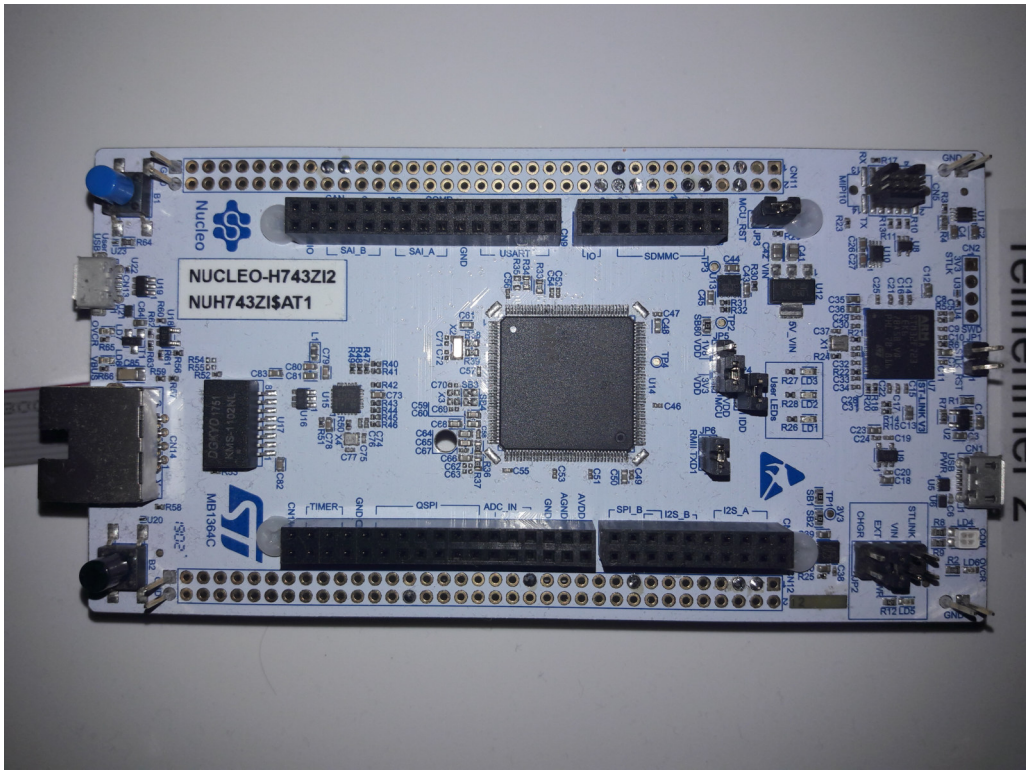


Abbildung 3.1.1: NUCLEO-Board STM32H743ZI2

Auswahlkriterien

Ausgewählt wurde dieses Board, aufgrund der nötigen Anforderungen für das Projekt. Durch den redundanten CAN-Bus muss ein Board mit zwei CAN (FD)-Controllern verwendet werden. Ebenso muss das Board eine I^2C-Schnittstelle und die passende Schnittstelle bzw. Pinout für das Display bereitstellen, ohne Doppelbelegungen der Pins zu verursachen. Hierzu werden außerdem Pins benötigt, die einen Analog Digital Converter (ADC) für die Bedienung des Touchdisplays bereitstellen können. Die maximale Taktfrequenz der CPU von 480 MHz ist ebenfalls sehr nützlich für eine schnellere Bildwiederholrate des Displays. [15]
Programmiert wird der Microcontroller mit Integrated Development Environment (IDE) von ST: STM32CubeIDE. Mithilfe des CubeMX-Assistenten wurden bereits in der graphischen Oberfläche die wichtigsten GPIO-Konfigurationen vorgenommen (siehe Tabelle 3.1.1):

Konfiguration	**Einstellung**
CPU-Clock	480 MHz
I^2C-Clock	75 MHz
CAN FD-Clock	75 MHz
ADC-Clock	75 MHz
Timer 1	Prescaler 479 (Timer mit 1 μs)
FDCAN1	Bitrate von 500/1000 MBit/s
FDCAN2	Bitrate von 500/1000 MBit/s (Bitrate-Switching)
I^2C	Standard-Einstellungen
FREERTOS	benötigte Tasks folgen in Kapitel 5

Tabelle 3.1.1: Vorgenommene Einstellungen in CubeMX

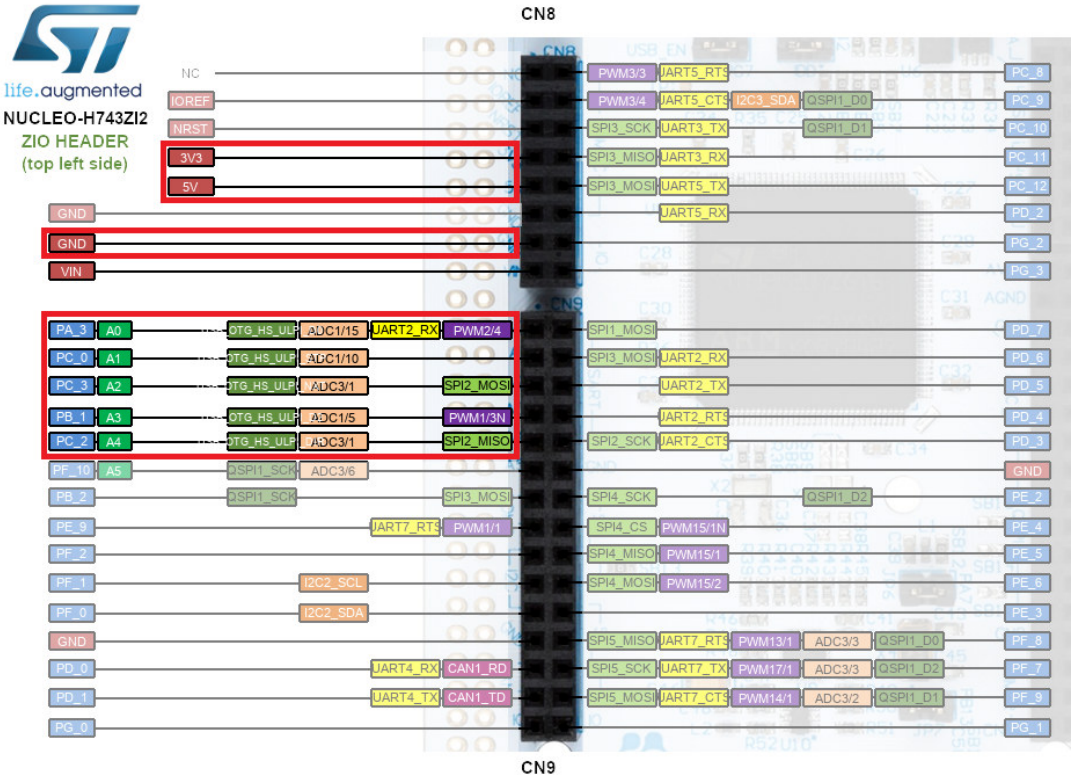


Abbildung 3.1.2: Nucleo STM32H7: Belegung der Aufsteck-Pinleiste links [15]

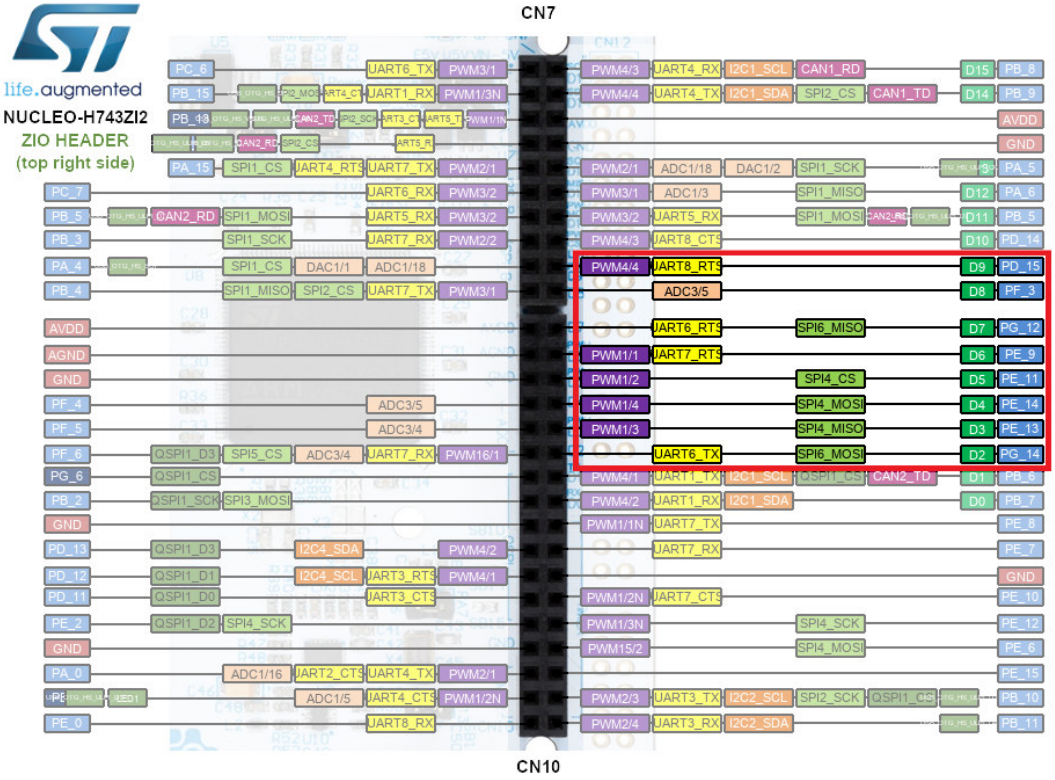


Abbildung 3.1.3: Nucleo STM32H7: Belegung der Aufsteck-Pinleiste rechts [15]

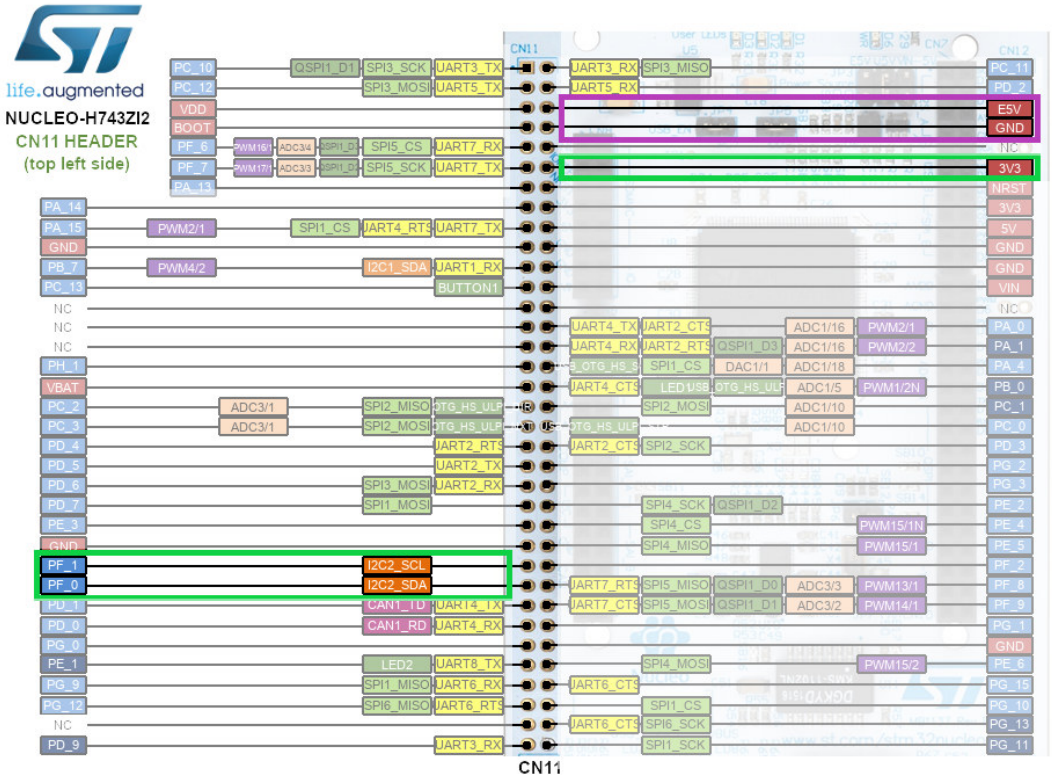


Abbildung 3.1.4: Nucleo STM32H7: Löt-Pinleiste links [15]

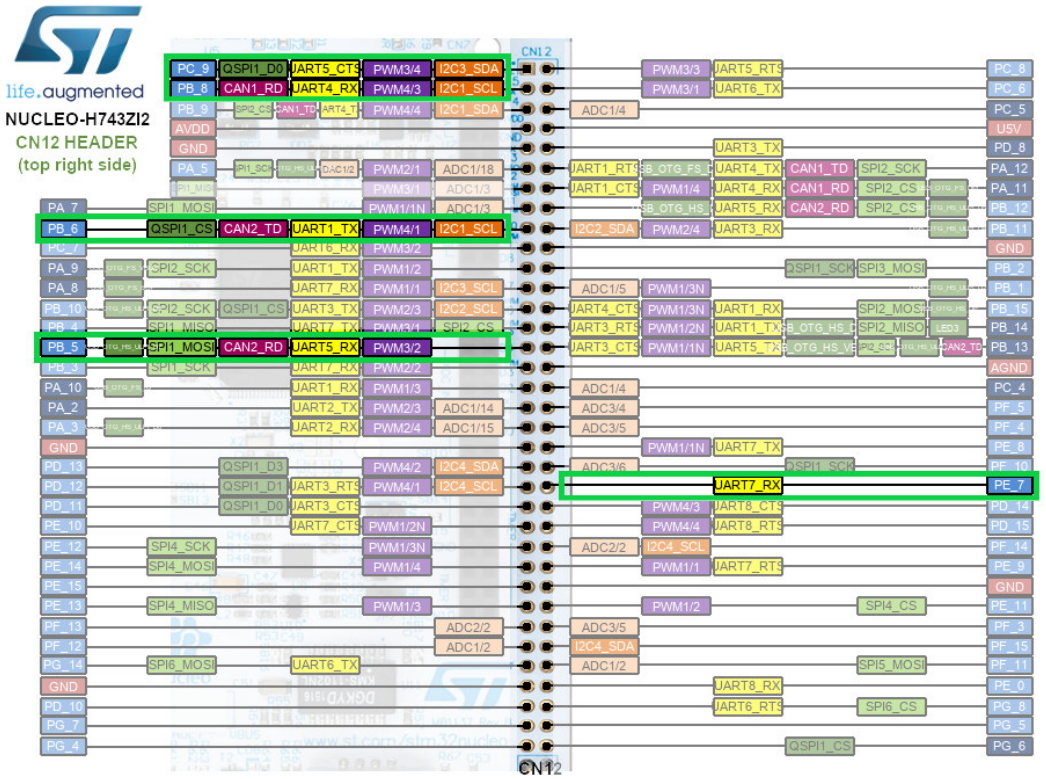


Abbildung 3.1.5: Nucleo STM32H7: Löt-Pinleiste rechts [15]

3.2 Bauteile des FIB-Aufbaus

Das FIB (Demonstratorplatine) übernimmt die Funktion des redundanten CAN-Netzwerks und der Fehlersimulation auf diesem Netzwerk. Es werden im Folgenden die verwendeten Bauteile näher beschrieben und auf deren Funktion eingegangen.

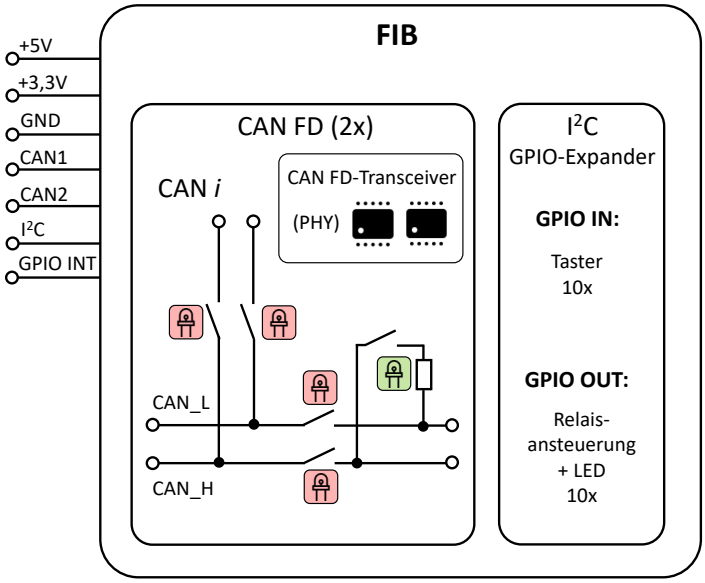


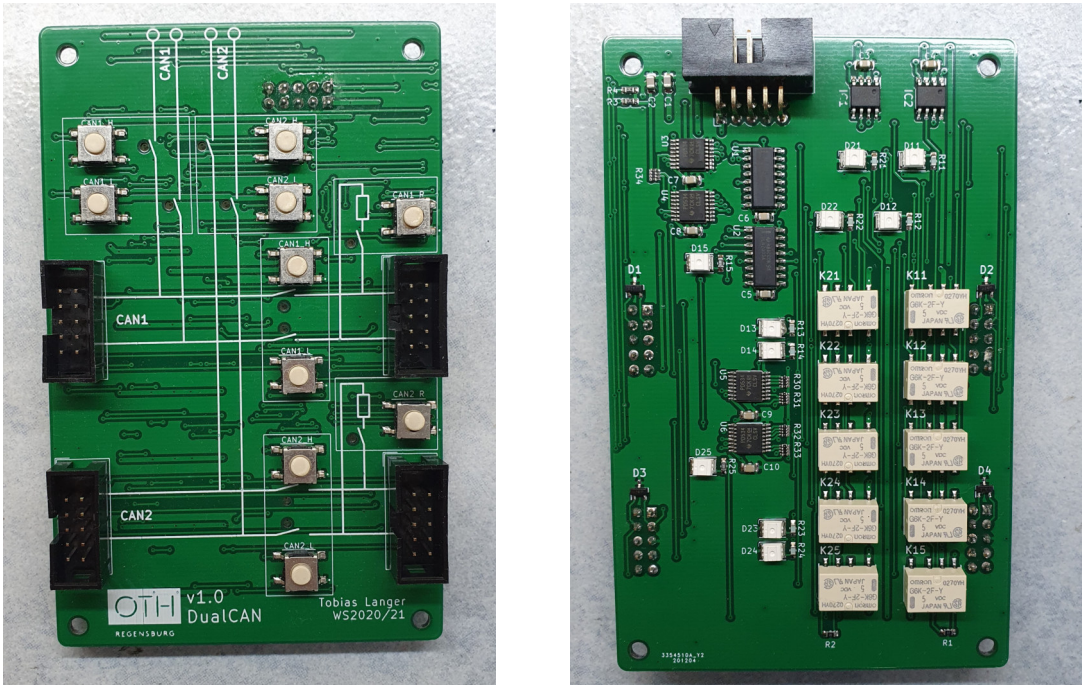
Abbildung 3.2.1: Blockschaltbild des FIB

In Abb. 3.2.1 ist der schematische Aufbau des FIB als Blockschaltbild dargestellt sowie das Einbringen der Unterbrechungen in CAN_L und CAN_H, dem Abschlusswiderstand und der LED-Statusanzeige (links). Rechts wird die Logik zur Abfrage der Taster und die LED-Ansteuerung dargestellt.
Die Platine wird mit den Spannungen +3,3 V und +5 V versorgt. Außerdem werden die beiden CAN-Busse und der I²C-Bus mit dem Microcontroller verbunden. Die GPIO-Erweiterungsbausteine können zudem einen Interrupt bei Änderung der Inputs auslösen. Durch die beiden CAN-Transceiver werden die Spannungen der Netzwerke auf die richtigen CAN-Pegel gehoben.
Durch die Taster am Board kann das entsprechende Relais und die zugehörige LED geschaltet werden und somit eine Unterbrechung im CAN-Netzwerk simuliert werden.

In der Tabelle 3.2.1 sind die verwendeten Pins des Microcontrollers durch das FIB aufgelistet.

Anschluss FIB	Pin STM32H7	Anschluss FIB	Pin STM32H7
+3,3V	3V3	GPIO_INT	PE_7
+5V	5V	FDCAN1_RX	PB_8
GND	GND	FDCAN1_TX	PB_9
I2C_SCL	PF_1	FDCAN2_RX	PB_5
I2C_SDA	PF_0	FDCAN2_TX	PB_6

Tabelle 3.2.1: Verwendete Löt-Pins des Nucleo STM32H7 durch das FIB



(a) (b)

Abbildung 3.2.2: Failure Injection Board: Ansicht von a) vorne und b) hinten

Das fertig bestückte FIB ist in Abb. 3.2.2 dargestellt. Auf der Oberseite (a) sind die Anschlüsse der CAN-Leitungen zu erkennen. Ebenso sind schematisch die Wege der Leitungen aufgezeichnet. Die Taster schalten die zugehörige Unterbrechung bzw. den Abschlusswiderstand. Die LEDs zeigen den Schaltzustand des jeweiligen Relais an.
Die Rückseite (b) zeigt den Anschluss zum Microcontroller sowie die beiden CAN-Transceiver, die GPIO-Erweiterungen, die Relais-Treiber, die Relais selbst und die LEDs, die durch ein Loch von der Rückseite leuchten.

3.2.1 CAN-Transceiver

Der CAN-Transceiver setzt die TTL-Signale des Microcontrollers auf die differentiellen Signale CAN_Low und CAN_High um. Außerdem wird hier eine galvanische Trennung vorgenommen zwischen den Signalpegeln. Die beiden CAN-Transceiver MCP2561FD von Microchip Technology sind mit der Versorgungsspannung +5 V verbunden und sind mit CAN FD kompatibel. Es werden je 2 Leitungen (Tx und Rx) vom Microcontroller (bzw. CAN-Controller) benötigt. [16]
In Abb. 3.2.3 ist die Verschaltung des CAN-Transceivers dargestellt.

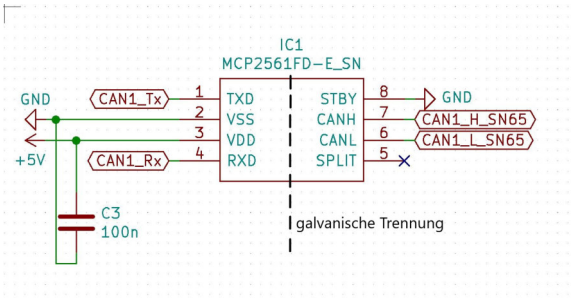


Abbildung 3.2.3: Schaltsymbol des CAN-Transceivers

Zum Brührungsschutz der CAN-Transceiver sind je zwei Electrostatic Discharge (ESD)-Dioden verbaut vom Typ NUP2105L des Herstellers ON Semiconductor. [17]

3.2.2 I²C und GPIO-Erweiterung

Um viele Adern und Leitungswege sowie GPIO-Pins des Microcontrollers zu sparen, werden auf dem FIB mehrere I²C-GPIO-Expander eingesetzt. Es werden dadurch nur 3 Leitungen vom Board zum Microcontroller benötigt (I2C_SDA, I2C_SCL und GPIO_INT).

I²C

I²C ist ein Bus mit Zweidrahtverbindung. Der Bus ist eine Master-Slave-Architektur konzipiert für kurze Distanzen zur Kommunikation von Controller und Sensoren oder anderen Integrated Circuit (IC)-Bausteinen. Neben den Versorgungsspannungsleitungen werden nur zwei weitere benötigt (SDA: Datenleitung, SCL: Taktleitung). Die Teilnehmer können gezielt den Bus auf einen Low-Pegel ziehen, da SDA und SCL im Idle-Zustand mittels Pull-Up-Widerstand auf High gesetzt werden. In Abb. 3.2.4 ist der schematische Aufbau des I²C-Bus dargestellt.
Das I²C-Bussystem erlaubt gleichzeitig mehrere Master die miteinander kommunizieren. Da bei dieser Anwendung eine klare Hierarchie zwischen Master und Slave besteht, kann in der gewünschten Betriebsart nur der Master Daten anfordern. Auf diese Anfrage sendet der Slave im Anschluss die Daten. [18]

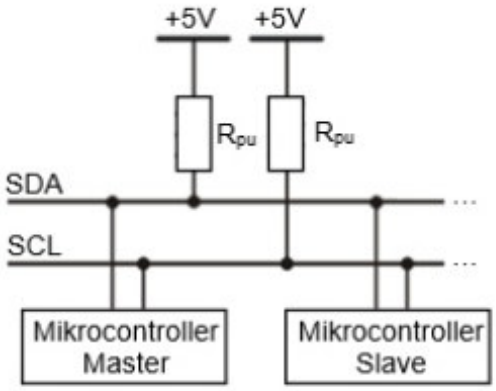


Abbildung 3.2.4: Aufbau des I2C Datenbus [19]

Die Adressierung der Slaves geschieht meist mit 7 Bits, d.h. es stehen 128 Adressen im Bus zur Verfügung, welche zum Teil aber reserviert sind. Durch das Setzen der jeweiligen Pins auf High oder Low werden die Adressen an den ICs eingestellt. Die Daten werden in 8 Bit Sequenzen gesendet (siehe Abb. 3.2.5). [19]

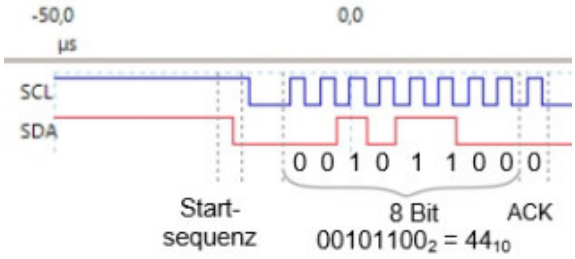


Abbildung 3.2.5: Übertragung von Daten beim I2C Datenbus [19]

GPIO-Erweiterung mit Port-Expander

Als GPIO-Erweiterungen wurden ICs des Herstellers Texas Instruments verwendet. Der Baustein PCA9534 ist eine 8 Bit Erweiterung der GPIO-Pins des Microcontrollers und kommuniziert über I^2C. Zudem wird eine externe Interrupt-Leitung zum Microcontroller geführt, sodass dieser bei einer Änderung der Taster unmittelbar reagieren kann. [20]
Die Adressen der GPIO-Erweiterungen sind in diesem Projekt nach folgender Tabelle 3.2.2 vergeben:

GPIO-Expander	In/Out	Adresse
CAN1 Relais	OUT	0x20
CAN2 Relais	OUT	0x21
CAN1 Taster	IN	0x22
CAN2 Taster	IN	0x23

Tabelle 3.2.2: Adressen der verwendeten GPIO-Erweiterungen auf dem FIB

Abb. 3.2.6 zeigt das Schaltsymbol der GPIO-Erweiterung mit der Adresse 0x20.

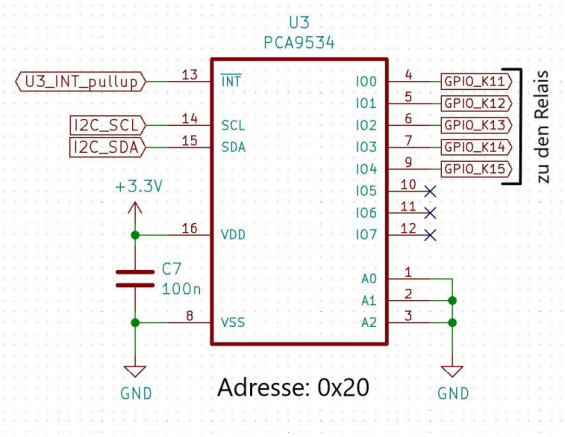


Abbildung 3.2.6: Schaltsymbol der GPIO-Erweiterung

3.2.3 Relais und Treiberbausteine zur Drahtbruchsimulation

Die Relais unterbrechen die CAN_H/CAN_L-Leitungen des jeweiligen CAN-Busses gezielt an der geforderten Stelle, um einen Drahtbruch zu simulieren. Mit dem zweiten Wechselkontakt des Relais wird eine Anzeige-LED geschaltet. Es wird somit eindeutig der Fehlerort angezeigt. Ebenso wird ein Relais zum Schalten der Abschlusswiderstände verwendet. In Abb. 3.2.7 ist das Schaltsymbol des Relais dargestellt. Verwendet wurden Relais G6K des Herstellers Omron. [21]

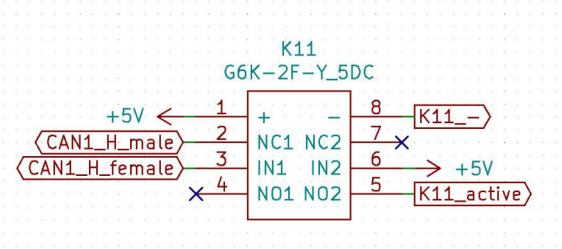


Abbildung 3.2.7: Schaltsymbol des Relais

Um die 5V-Relais mit den GPIO-Ausgängen schalten zu können wird ein weiterer Baustein benötigt. Hierzu wird ein ULN2003 von ST (Schaltsymbol siehe Abb. 3.2.8) verwendet. Es wird der Spannungspegel angepasst und der benötigte Strom des Relais kann geliefert werden. Außerdem verhindert der Relais-Treiber IC eventuelle Spannungsspitzen durch das Schalten der Spule im Relais am GPIO-Kontakt. [22]

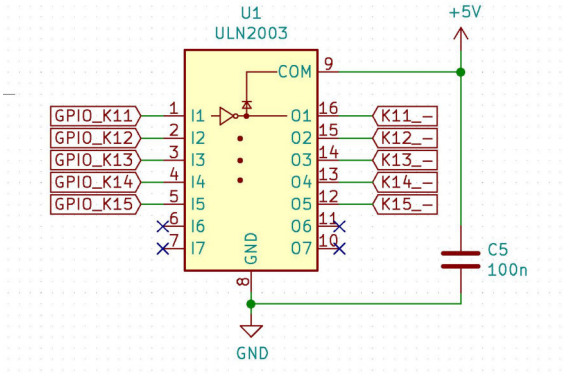


Abbildung 3.2.8: Schaltsymbol des Relais-Treiber ICs

Abb. 3.2.9 zeigt die prinzipielle Funktionsweise des Port-Expanders und die Relaisansteuerung mittels Treiberbausteinen. Der Schließerkontakt des Relais schaltet die LED, der Öffner simuliert den Drahtbruch im System (für einen Abschlusswiderstand wird ebenfalls ein Schließer verwendet).

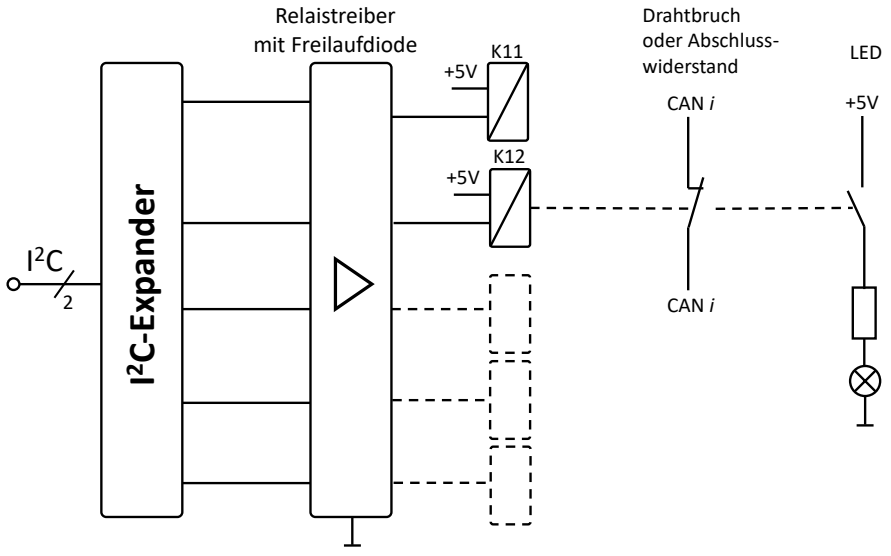


Abbildung 3.2.9: Zusammenarbeit von GPIO-Expander, Relaistreiber und die Schaltkontakte des Relais

3.2.4 Ansteuerung und Anzeige

In Abb. 3.2.10 ist beispielhaft eine Unterbrechung beider Leitungen im CAN2 geschaltet. Die Unterbrechungen sind durch die Taster auf dem FIB zu schalten oder auch durch das später erwähnte Touchdisplay (siehe Kapitel 4). [23]
Durch die Bauform der LEDs ist eine rückseitige Montage auf der Platine möglich. Sie leuchten durch eine Bohrung auf die Oberseite der Platine und zeigen so die unterbrochene Stelle im Bus an. Verwendet wurden hier LEDs der Firma AGILENT mit dem Footprint PLCC-2 [24]. Die Vorwiderstände wurden passend zu den roten bzw. den grünen LEDs berechnet [25].

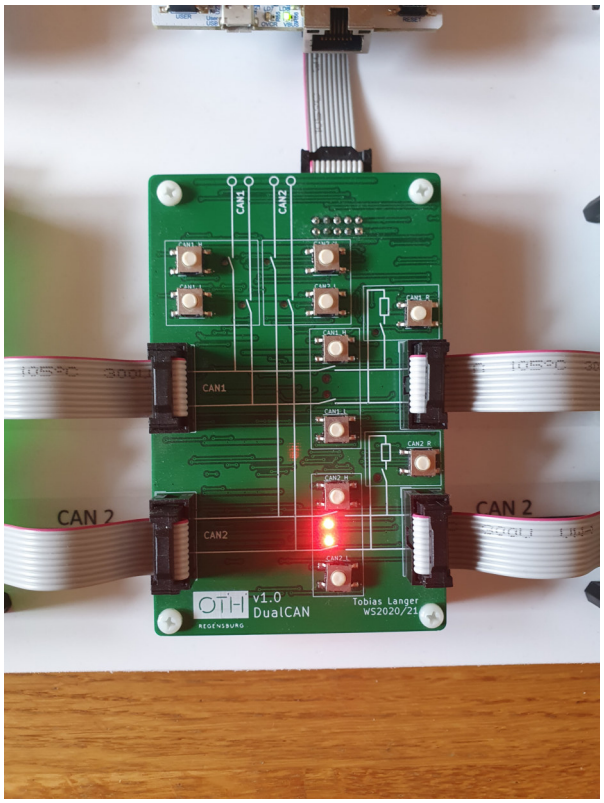


Abbildung 3.2.10: Failure Injection Board: LED-Anzeige

3.2.5 Schaltbare Abschlusswiderstände

Um eine ordnungsgemäße Übertragung bei CAN gewährleisten zu können sind an den beiden letzten Teilnehmern des Busses Abschlusswiderstände nötig. Die Widerstände von 120 Ω sind hier jeweils durch Relais hinzu- bzw. wegzuschalten. Abb. 3.2.11 zeigt das letzte FIB des Linienbussystems mit den aktivierten Abschlusswiderständen für CAN1 und CAN2. Ebenfalls ist hier bereits im Display auch die Anzeige der aktivierten Widerstände zu sehen.

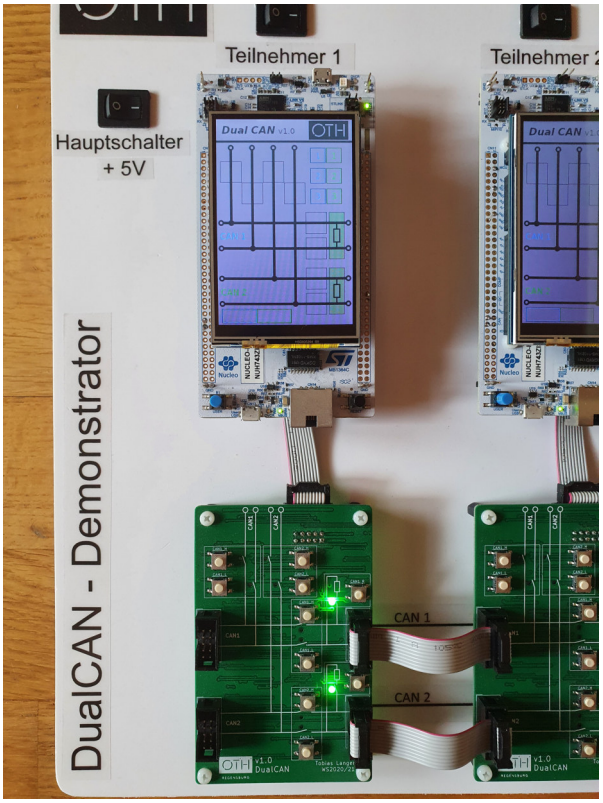


Abbildung 3.2.11: Failure Injection Board: Abschlusswiderstände

3.3 Platinendesign

Das FIB wurde mit dem OpenSource Programm KiCad entworfen und gezeichnet. Im Anhang befindet sich der gezeichnete Schaltplan mit allen verwendeten Bauteilen. Dieser Schaltplan ist noch nicht ganz korrekt. In der ersten Version wurden die Wege und Verläufe des CAN-Bus über die Relais nicht korrekt gezeichnet (mehr hierzu in Kapitel 7.1).
Die Stecker zur Verbindung mit den anderen FIB sind auf der Oberseite zu finden. Die Verbindung zum Microcontroller sitzt auf der Unterseite. Die Platine wurde zudem so designed, dass die Bedien- und Anzeigeelemente auf der Oberseite zu finden sind sowie die schematisch dargestellten Wege der Busse. Auf der Rückseite sitzen alle weiteren benötigten Bausteine. Das Platinenlayout sowie die Leitungswege und Footprints sind in Abb. 3.3.1 (Layer oben) und 3.3.2 (Layer unten) dargestellt. Die Maße der Platine sind 70x100 mm.

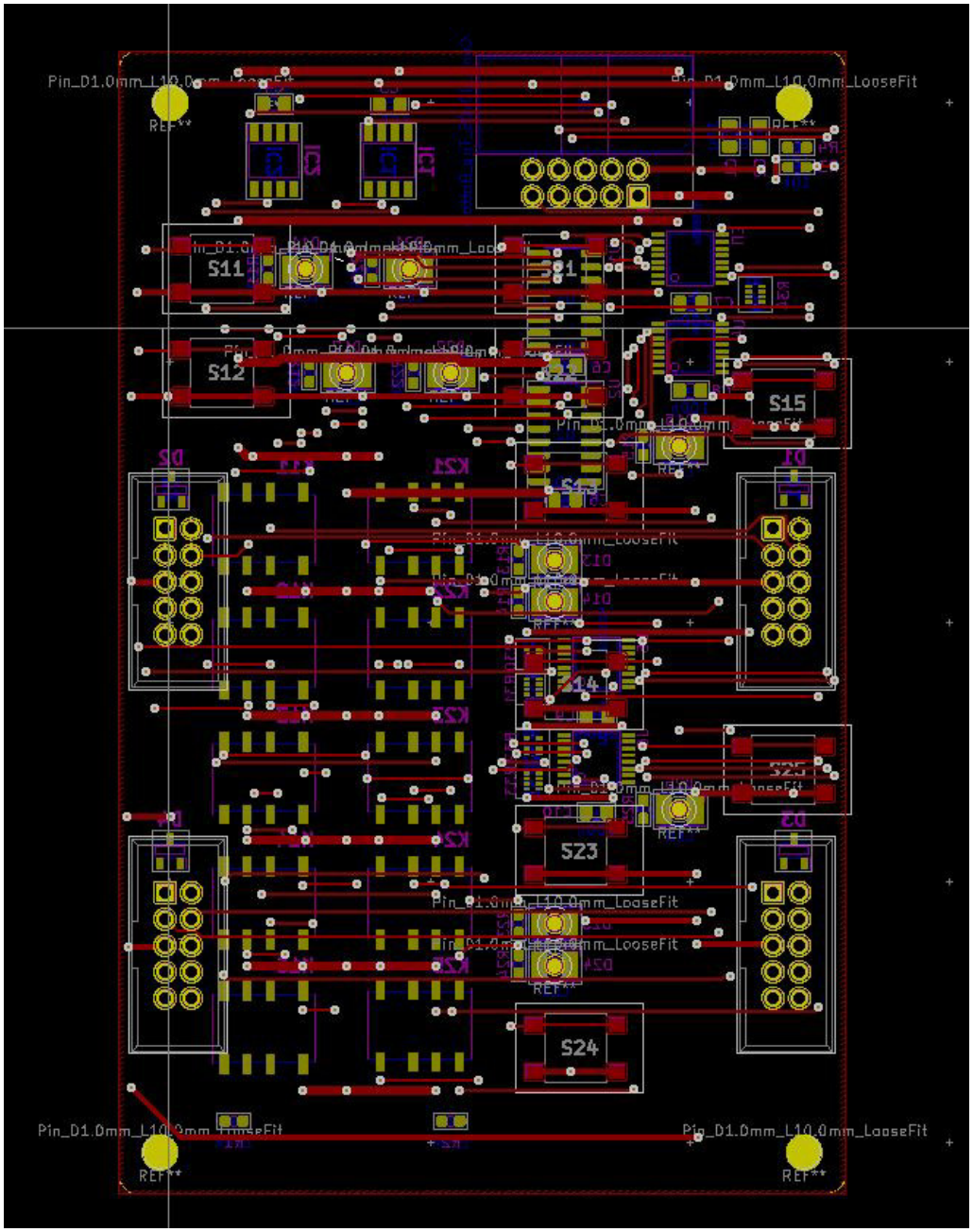


Abbildung 3.3.1: Failure Injection Board: Platinendesign Layer oben

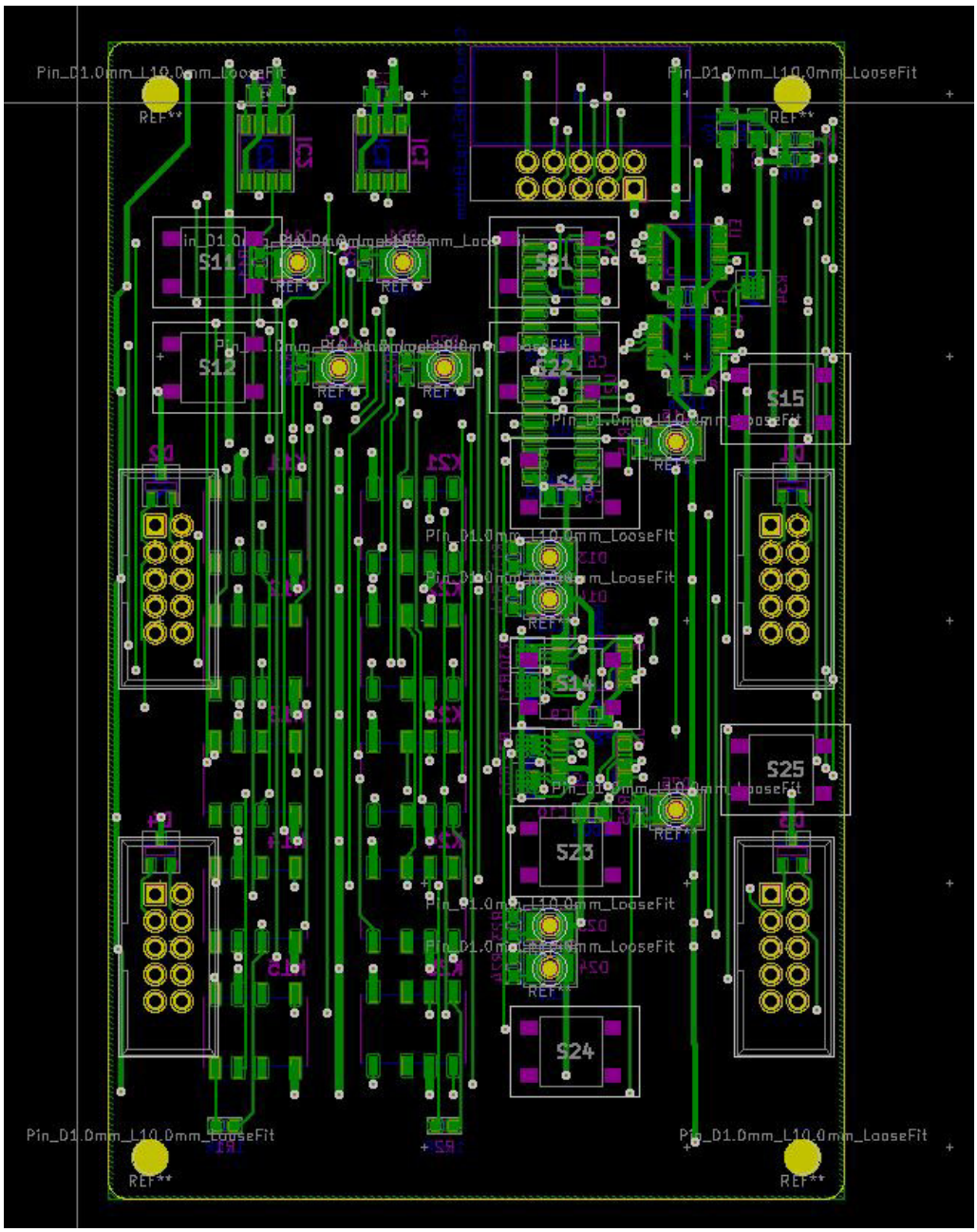


Abbildung 3.3.2: Failure Injection Board: Platinendesign Layer unten

4. Display: Anzeige und Steuerung

Zur besseren Visualisierung und komfortablen Bedienung des Dual-CAN Demonstators wird zusätzlich zu den LEDs und den Tastern auf dem FIB ein Touchdisplay (TFT-Liquid Crystal Display (LCD)) auf dem Nucleo Board angebracht. Hier soll die Fehlerinjektion sowie auch einige Einstellungen vorgenommen und CAN-Nachrichten gesendet werden können.

4.1 TFT Display

Für diese Anwendung bietet sich ein Touchdisplay in der Größe von 3,5" an. Dieses Display des Herstellers Kuman (siehe Abb. 4.1.1) kann mit einer Auflösung von 480x320 Pixel Grafiken (bspw. Bitmap-Dateien) mit 65K Farbtiefe ausgeben und diese in einer mehr als ausreichenden Qualität für diese Anwendung darstellen. Die Ansteuerung wird über einen 8-Bit Parallelbus realisiert und ist damit deutlich schneller als vergleichbare Displays mit einer I^2C- oder Serial Peripheral Interface (SPI)-Schnittstelle. Mittels dem Treiberbaustein ILI9486 werden die Daten auf dem Display verarbeitet und ausgegeben.
Zusätzlich ist auch ein SD-Karten-Slot verbaut, um große Bilddateien lesen und ausgeben zu können. Dies funktioniert über einen SPI-Bus, der unabhängig von der Display-Ansteuerung verdrahtet werden muss (diese Funktion wird in dieser Arbeit nicht verwendet). [26, 27]

(a) (b)

Abbildung 4.1.1: Touchdisplay: Ansicht von oben und unten

Die Touchfunktion wird durch Analogwertmessung realisiert. Hierzu muss die Spannungsänderung in x- bzw. y-Richtung gemessen und ausgewertet werden.
Abb. 4.1.2 zeigt die nötigen Verbindungen, wie den Datenbus und die Kontrollleitungen, zwischen dem Evaluierungsboard und dem Touchdisplay.

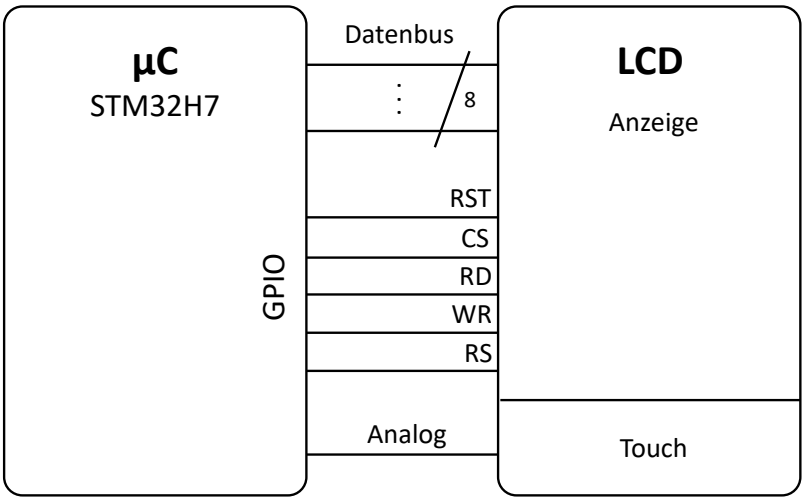


Abbildung 4.1.2: Verbindungen zwischen dem STM32H7 Board und dem Touchdisplay

4.1.1 Anzeige von Text und Grafiken

Als Grundlage zur Programmierung der Ausgabe des Touchdisplays auf dem Nucleo STM32H7 wurde die Arduino-Bibliothek Adafruit-GFX-Library und Mcufriend_kbv verwendet. Die Anpassungen der Bibliothek an den STM32 Controller wurde durch ControllersTech vorgenommen [28, 29, 30].
Zur Programmierung wird auch hier die STM32CubeIDE verwendet. Zu Beginn werden die Ports wie in nachfolgender Tabelle 4.1.1 verbunden bzw. das Display auf die Pinleiste gesteckt und die Pinkonfiguration in CubeMX vorgenommen (siehe Abb. 4.1.3).

Pin Display	Pin STM32H7	Pin Display	Pin STM32H7
LCD_3V3	3V3	LCD_D2	PG_14
LCD_5V	5V	LCD_D3	PE_13
LCD_GND	GND	LCD_D4	PE_14
LCD_RST	PC_2	LCD_D5	PE_11
LCD_CS	PB_1	LCD_D6	PE_9
LCD_RS	PC_3	LCD_D7	PG_12
LCD_WR	PC_0	SD_SS*	PD_14*
LCD_RD	PA_3	SD_DI*	PB_5*
LCD_D0	PF_3	SD_DO*	PA_6*
LCD_D1	PD_15	SD_SCK*	PA_5*

Tabelle 4.1.1: Verwendete Pins des Displays auf dem Nucleo STM32H7 (: nicht verwendet)*

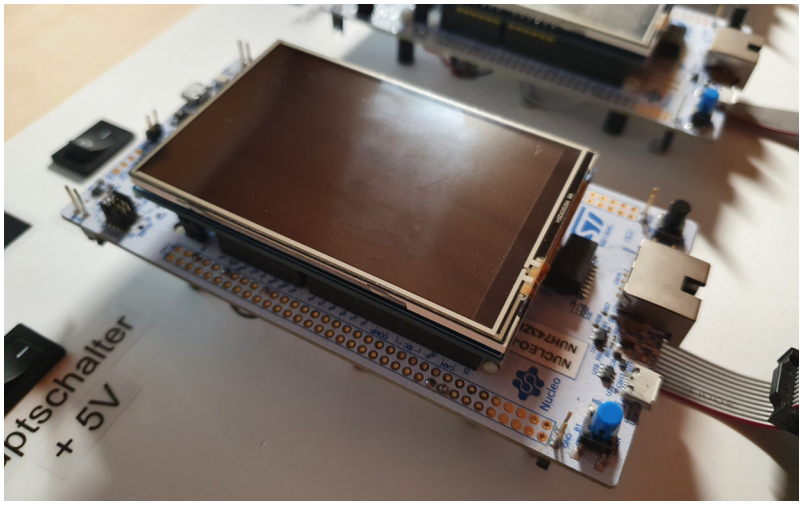


Abbildung 4.1.3: Display aufgesetzt auf dem Nucleo STM32H7 Board

Portmapping

In der Bibliothek von ControllersTech müssen einige Anpassungen vorgenommen werden, um das Display verwenden zu können. Wie der Tabelle 4.1.1 zu entnehmen ist, müssen die jeweiligen Pins im Programm abgeändert und angepasst werden sowie der Support für die Displayart 9488_555 (*#define SUPPORT_9488_555*) aktiviert werden. Außerdem werden in der Datei *user_settings.h* folgende Änderung bei der *write_8(data)*- und *read_8()*-Funktion vorgenommen [30]:

```
#define write_8(d) {GPIOF->BSRR = 0b0000000000001000 << 16; \
                    GPIOD->BSRR = 0b1000000000000000 << 16; \
                    GPIOG->BSRR = 0b0101000000000000 << 16; \
                    GPIOE->BSRR = 0b0110101000000000 << 16; \
                    GPIOF->BSRR = (((d) & (1<<0)) << 3);    \
                    GPIOD->BSRR = (((d) & (1<<1)) << 14);   \
                    GPIOG->BSRR = (((d) & (1<<2)) << 12)    \
                                | (((d) & (1<<7)) << 5);    \
                    GPIOE->BSRR = (((d) & (1<<3)) << 10)    \
                                | (((d) & (1<<4)) << 10)    \
                                | (((d) & (1<<5)) << 6)     \
                                | (((d) & (1<<6)) << 3);    \ }
#define read_8() (        (((GPIOF->IDR & (1<<3))  >> 3) \
                          | ((GPIOD->IDR & (1<<15)) >> 14)\
                          | ((GPIOG->IDR & (1<<14)) >> 12)\
                          | ((GPIOE->IDR & (1<<13)) >> 10)\
                          | ((GPIOE->IDR & (1<<14)) >> 10)\
                          | ((GPIOE->IDR & (1<<11)) >> 6) \
                          | ((GPIOE->IDR & (1<<9))  >> 3) \
                          | ((GPIOG->IDR & (1<<12)) >> 5)))
```

Listing 4.1: Änderung in der Sende- und Empfangsfunktion

weitere Anpassungen

Da nun die Schreibe- und Lesefunktion auf die verwendeten Ports angepasst ist, muss zudem ein Timer konfiguriert werden, der im μs-Takt zählt. Hierzu wird in CubeMX der Timer 1 (TIM1) konfiguriert. Es wurde die maximale Taktfrequenz der CPU von 480 MHz gewählt, da so die LCD-Ansteuerung am schnellsten realisiert wird. Der Timer wird mit einem Prescaler von 479 und einer Counter Period von 0xFFFE initialisiert. [31]

Funktionen zur Bildausgabe

In der gegebenen Bibliothek sind mehrere Funktionen vordefiniert, um geometrische Formen wie Kreise, Rechtecke oder gerade Linien auszugeben. Ebenso kann auch direkt ein String als Text auf dem Display ausgegeben werden. Alle Funktionen beruhen auf dem gleichen Prinzip, die Daten über den 8-Bit Parallelbus an das Display zu übergeben.
Als Grundlage, um Daten *writeData(x)* bzw. Kontrollbefehle *writeCMD(x)* senden zu können, wird die *write_8(x)* Funktion aufgerufen, die 8 Datenbits parallel senden kann. Durch das Setzen des Pins RS (Register Select) auf Low wird ein Kommandobefehl gekennzeichnet. Während des gesamten Sendevorgangs muss der Pin CS (Chip Select) auf Low gesetzt werden. Erst nach Beenden des kompletten Sendevorgangs wird dieser wieder auf High gesetzt. Das genaue Timing ist dem Handbuch [26] zu

entnehmen (siehe Abb. 4.1.4).
Als ersten Schritt wird mithilfe der Funktion *setAddrWindow(x, y, x1, y1)* das zu beschreibende Pixelfenster (oder auch ein einzelnes Pixel) an das Display übergeben. Hierbei wird dem Display mithilfe der Kontrollbefehle die jeweilige Start- und Endspalte bzw. -zeile übergeben (siehe Tabelle 4.1.2).

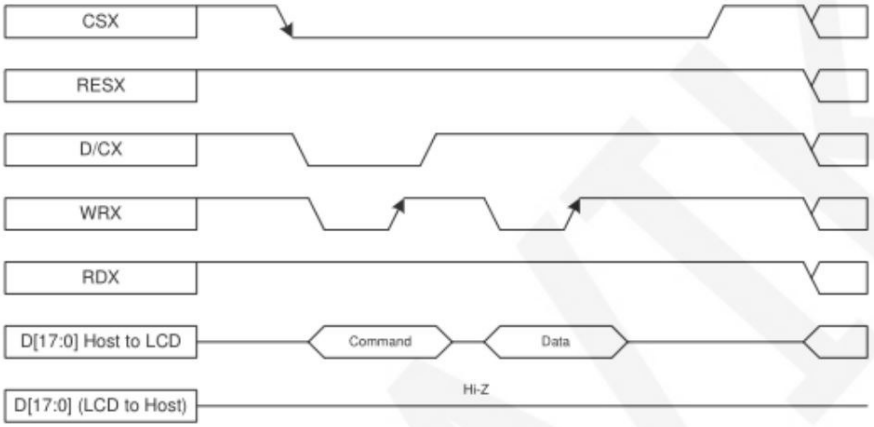


Abbildung 4.1.4: Timingdiagramm der write Funktion [26]

Kontrollbefehl	**Beschreibung**	**Daten**
_SC = 0x2A	Column Address Set	x(MSB, LSB), x1(MSB, LSB)
_SP = 0x2B	Page Address Set	y(MSB, LSB), y1(MSB, LSB)

Tabelle 4.1.2: Beispiele für wichtige Kontrollbefehle zum Senden von Daten (Setzen der aktuellen Pixelfensters) [32]

Abschließend werden die Daten der Pixel byteweise übergeben. Für jedes Pixel werden 16-Bit-Farbwerte gefordert. Diese Werte müssen im 565-Modus abgespeichert werden. Das heißt die 16 Bit des Pixels werden nach dem Farbcode, wie in Abb. 4.1.5 zu erkennen ist, dargestellt.
Nachdem ein 8-Bit Parallelbus verwendet wird, werden 2 Sendevorgänge pro Pixel benötigt (exklusive den Kontrollbefehlen). In der ursprünglichen *drawBitmap()*-Funktion wird für jedes Pixel das Fenster für das zu übertragende Pixel neu gesetzt. Dadurch wird im Vergleich zur gewünschten Reaktionszeit des Displays sehr viel Zeit verschwendet.

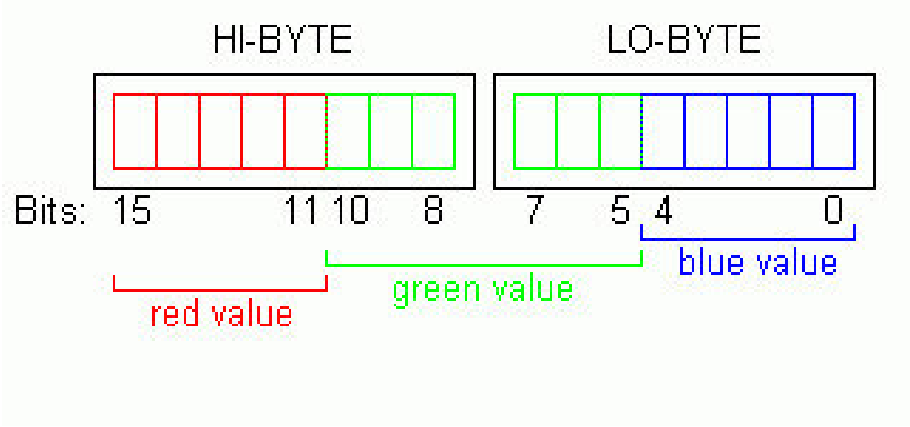


Abbildung 4.1.5: Farbcode für 565-codierte Pixel [33]

drawRGBBitmap()

Für jede Grafik, die auf dem Display angezeigt werden soll, muss ein Array mit den jeweiligen Pixeldaten angelegt werden. So besitzt eine Bitmapdatei mit 50x50 Pixeln 250 Arrayeinträge mit je 16-Bit Werten (500 Byte).

```
1  void drawRGBBitmap(uint16_t *bitmap, int16_t x, int16_t y, int16_t
       w, int16_t h)
2  {
3    int32_t n;
4    n = w*h;
5    setAddrWindow(x, y, x+w-1, y+h-1);
6      CS_ACTIVE;
7      WriteCmd(0x2C);
8
9    for(int i=0; i < n; i++)
10   {
11     WriteData(bitmap[i]);
12   }
13     CS_IDLE;
14 }
```

Listing 4.2: drawRGBBitmap()-Funktion

Der Funktion *drawRGBBitmap()* wird nun das Array mit den Pixeldaten übergeben sowie die Startkoordinaten und die Größe des Bildes. Die Bitmap-Dateien werden mithilfe einer ausführbaren Datei passend für die Anwendung in Arrays umgewandelt. Die Ausgabe der .exe-Datei befindet sich in einer Textdatei. Diese kann kopiert werden und so in den Programmcode eingefügt werden. In Abhängigkeit dieses Arrays wird nun das zu beschreibende Pixelfenster mithilfe der richtigen Kommandobefehle gesetzt.

Im Anschluss beginnt der Sendevorgang der Daten, der durch das Kommando 0x2C eingeleitet wird. Nach Abschluss des Sendevorgangs wird *CS_IDLE* wieder gesetzt und dem Display mitgeteilt, dass alle Daten gesendet wurden.

printnewstr()

Um einen Text direkt auf dem Display auszugeben werden in der Adafruit-GFX-Library bereits einige Schriftarten sowie -größen zur Verfügung gestellt. [28]
Der Funktion wird der Startwert des Textes in Form von Koordinaten übergeben. Auch die Schriftart selbst, welche nach dem ASCII-Code für jedes Schriftzeichen einen Arrayeintrag beinhaltet, die Textfarbe und -größe werden der Funktion übergeben. Als wichtigsten Wert wird natürlich auch der String selbst, der den Text beinhaltet, übergeben.
Nachdem alle Randbedingungen gesetzt wurden, werden mit der *write()*-Funktion die Zeichen des Strings nacheinander ausgegeben und an das Display geschickt.

```
1 void printnewtstr (int row, int column, uint16_t txtcolor, const
    GFXfont *f, uint8_t txtsize, uint8_t *str)
2 {
3   setFont(f);
4   textcolor = txtcolor;
5   textsize = (txtsize > 0) ? txtsize : 1;
6   setCursor(column, row);
7   while (*str) write (*str++);
8 }
```

Listing 4.3: printnewstr()-Funktion

4.2 Touchscreen

Um die Bedienung per Touch zu ermöglichen, muss der Programmcode erweitert werden. Das verwendete Display besitzt ein resistives Touchfeld. Ein resistiver Touchscreen funktioniert über den Druck des Stifts oder Fingers auf die Oberfläche des Displays. Der Touchscreen besteht aus zwei leitfähigen Schichten, die durch den Finger oder einem Stift miteinander verbunden werden (siehe Abb. 4.2.1). Durch den entstandenen elektrischen Widerstand bzw. die davon abhängige Spannung, kann die Position berechnet werden. [34]

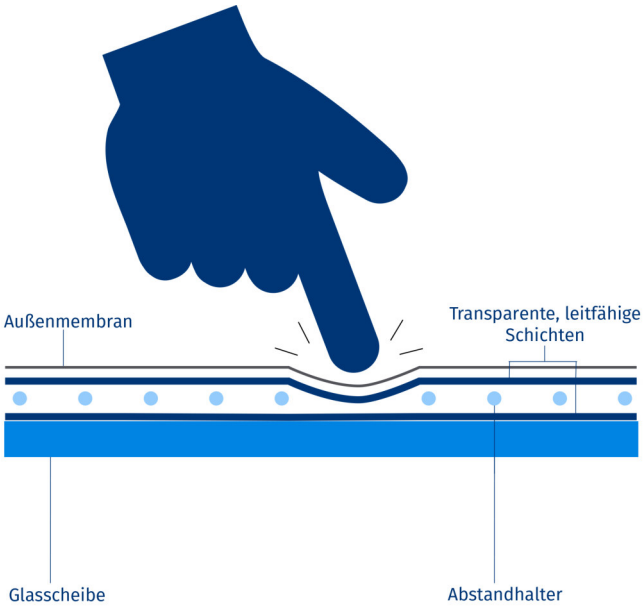


Abbildung 4.2.1: Aufbau eines resistiven Touchscreens [34]

4.2.1 Eingabe und Steuerung mittels Touchscreen

Durch Berührung auf dem Display verändern sich die Spannungen in x- bzw. y-Richtung und geben die Koordinaten des Druckpunkts wieder. Es kann ebenfalls die Stärke der Berührung errechnet werden. Diese wird in einer z-Komponente gespeichert und gibt an, ob eine Berührung stattfindet oder nicht.

In Tabelle 4.2.1 sind die für die Touchfunktion verwendeten Displaypins aufgelistet. Alle Pins werden nur umdefiniert und bleiben, wie bei der Datenübertragung, mit dem Microcontroller verbunden. Es wurde die Arduino-Bibliothek TouchScreen von Adafruit verwendet und für den STM32 Microcontroller entsprechend umgeschrieben [35].

Pin Display	Pin Touch	Definition
LCD_D0	LCD_YM	y-Koordinate Minus
LCD_D1	LCD_XP	x-Koordinate Plus
LCD_RS	LCD_XM	x-Koordinate Minus
LCD_CS	LCD_YP	y-Koordinate Plus

Tabelle 4.2.1: Verwendete Pins zur Touchfunktion und deren Definition [35]

Um die Koordinaten im Programmablauf auslesen zu können, müssen die GPIO-Pins zwischenzeitlich umdefiniert werden. In Tabelle 4.2.2 sind die jeweiligen Definitionen gegeben, um die x- bzw. y-Koordinate auslesen zu können. Wie eingangs erwähnt, müssen zwei Pins als ADC-Eingang im *Single-ended*-Modus konfiguriert werden, da Spannungswerte gemessen werden.

Pin Touch	x-Koordinate	y-Koordinate
LCD_YM	–	OUTPUT_HIGH
LCD_XP	OUTPUT_HIGH	–
LCD_XM	OUTPUT_LOW	ANALOG_IN
LCD_YP	ANALOG_IN	OUTPUT_HIGH

Tabelle 4.2.2: GPIO-Konfiguration für die Messung der jeweiligen Koordinaten [35]

Die Pins werden nach obiger Konfiguration (siehe Tabelle 4.2.2) definiert und können die x- und y-Koordinaten mittels Analog-Digital-Umwandlung auslesen. Die gemessenen Spannungswerte müssen anschließend noch den passenden Pixelwerten zugeordnet bzw. skaliert werden. Mithilfe der *map()*-Funktion können die vom ADC gelieferten Werte auf die Displaygröße von 320 Pixeln in x-Richtung sowie 480 Pixeln in y-Richtung skaliert werden. So kann nun mit den Pixelwerten gearbeitet werden und eine entsprechende Graphical User Interface (GUI) oder auch eine einfache Bedienoberfläche entwickelt werden.
Aufgrund von Bauteiltoleranzen muss diese Skalierung bzw. Kalibrierung für jedes Display neu vorgenommen werden.

5. Gesamtsystem

In diesem Kapitel wird der Programmablauf sowie die Gesamtfunktion des Demonstratoraufbaus genauer beschrieben. Zu Beginn wird die Bedien- und Anzeigeeinheit des FIB erläutert. Im Anschluss steht die Programmierung des Aufbaus im Fokus.

5.1 Funktion und Bedienung der Fehlerinjektion

Die Bedienung des FIB gestaltet sich recht einfach und intuitiv. Die Wege der beiden CAN-Netzwerke sind auf dem Board aufgezeichnet und schematisch mit Schaltkontakten an den Stellen gekennzeichnet, welche eine Unterbrechung ermöglichen.
Es gibt für jedes CAN-Netzwerk je zwei mögliche Unterbrechungen für die Low- bzw. High-Leitung. Die erste Schaltmöglichkeit ist eine Unterbrechung in der Stichleitung vom Microcontroller zur Hauptleitung (man schaltet nur den betroffenen Teilnehmer vom Netz, der restliche Bus bleibt funktionsfähig). Die andere Möglichkeit ist eine Unterbrechung zwischen zwei Teilnehmern. Wenn die Low- und High-Leitung geschaltet wird, wird der komplette Bus unterbrochen und in zwei Teile separiert. Beide Arten der Fehlerinjektion simulieren Drahtbrüche oder ähnliche Fehlersituationen, in welchen die Buskommunikation ausfällt. Diese Fehler sind farblich mit roten LEDs an den betroffenen Fehlerstellen markiert (siehe Abb. 5.1.1).
Weiterhin besteht die Möglichkeit an den letzten Teilnehmern des Demonstrators Abschlusswiderstände bei beiden CAN-Netzwerken hinzu- bzw. wegzuschalten. Um dies von den Fehlern abzuheben, ist ein aktivierter Abschlusswiderstand mit einer grünen LED markiert.
Weiterhin ist es auch möglich die Fehlerinjektion bzw. das Schalten von Abschlusswiderständen über das Touchdisplay vorzunehmen. Auf dem Display ist eine ähnliche Zeichnung des Netzwerkaufbaus zu sehen. Die aktivierten Stellen sind ebenso farblich markiert und durch Berührung steuerbar. Zusätzlich ist es hier möglich CAN-Nachrichten mit unterschiedlichen Daten zu senden. Die zuletzt empfangenen Nachrichten werden am unteren Bildschirmrand angezeigt.

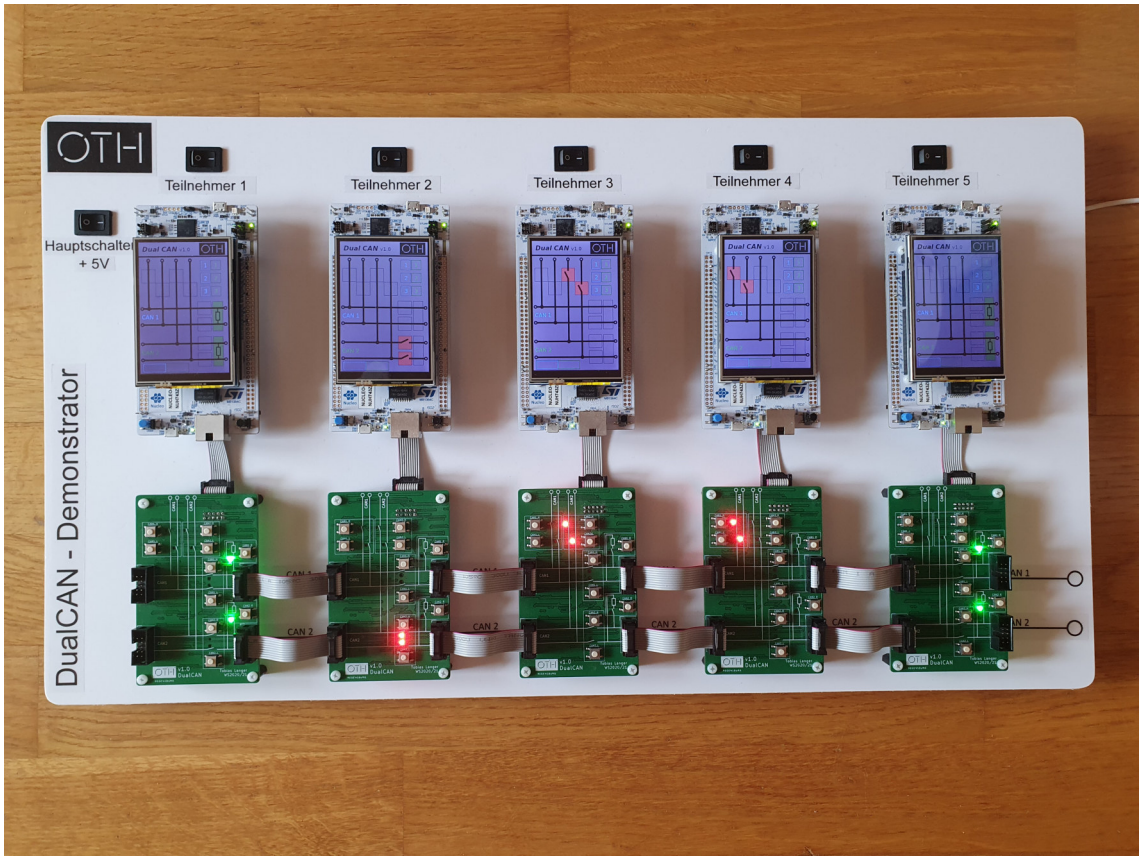


Abbildung 5.1.1: Gesamtaufbau mit geschalteten Fehlern und Abschlusswiderständen

5.2 Programmierung

Die grundsätzliche Funktionalität des Demonstratoraufbaus beruht auf mehreren Tasks. Der Task *Heartbeat* lässt eine LED im bestimmten Task ein- bzw. ausschalten, um anzuzeigen, dass der Microcontroller noch reagiert.
Drei weitere Tasks formen die Funktion des Aufbaus und kombinieren die Anzeige auf Touchdisplay, die Touchfunktion sowie die CAN-Kommunikation. Die Funktion des FIB wird über Interrupts implementiert.
In Abb. 5.2.1 ist das Flussdiagramm des Initialisierungsprozesses dargestellt. Es werden alle nötigen Peripherieports initialisiert sowie die Grundeinstellungen der Bussysteme und auch einzelner Funktionen vorgenommen. Nach der Initialisierung verweilt das Hauptprogramm in einer Endlosschleife und die Tasks übernehmen den Programmablauf.

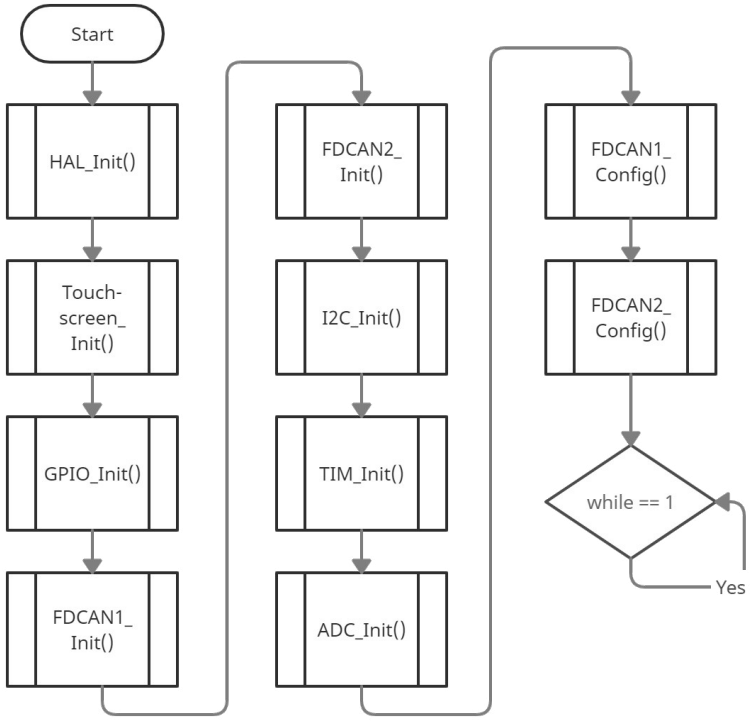


Abbildung 5.2.1: Flussdiagramm - Initialisierungsprozess für alle verwendeten Funktionen

5.2.1 Ansteuerung Failure Injection Board

Bei Programmbeginn wird der I^2C-Bus initialisiert. Den GPIO-Expandern wird ein In- bzw. Outputverhalten zugewiesen sowie Initialisierungswerte übergeben. Ebenso wird bei Start des Programms ein Selbsttest durchlaufen, in welchem jedes Relais ein- und ausgeschaltet wird.
Das FIB und die zugehörigen Relais werden über die GPIO-Expander geschaltet. Aufgrund einer Signaländerung eines Tasters wird ein Interrupt-Signal an den Microcontroller gesendet. Durch dieses Signal wird der Interrupt-Handler (siehe Abb. 5.2.2) aufgerufen. Hier wird anschließend der aktuelle Wert der Inputs ausgelesen und weiterverarbeitet. In Abhängigkeit davon wird den Output-Ports das jeweilige Signal wieder gesendet.

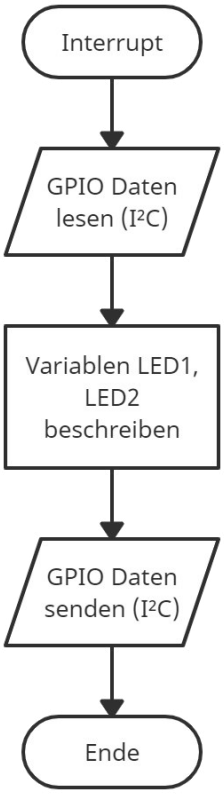


Abbildung 5.2.2: Flussdiagramm - Handler für GPIO-I2C-Expander

5.2.2 CAN-Kommunikation

Der Task *CAN* (siehe Flussdiagramm in Abb. 5.2.3) übernimmt die Aufgabe der Kommunikation über die beiden konfigurierten CAN-Busse. Es wird in einer Dauerschleife überprüft, ob eine neue Nachricht in der FIFO empfangen wurde. Wenn dies der Fall ist, wird diese auf Variablen übertragen.
Um schnell und eindeutig anzuzeigen, ob eine Testnachricht empfangen wurde, wird je eine LED (rot und orange) bei Empfang der Nachricht 0x80 geschaltet. Ebenso ist es möglich durch Betätigen des blauen Tasters auf dem Evaluierungsboard diese Testnachricht auf beiden Bussen zu senden.
Auf dem Touchdisplay ist es auch möglich den Befehl zum Senden von Testnachrichten zu geben. Auch diese Aufgabe wird vom Task *CAN* übernommen.

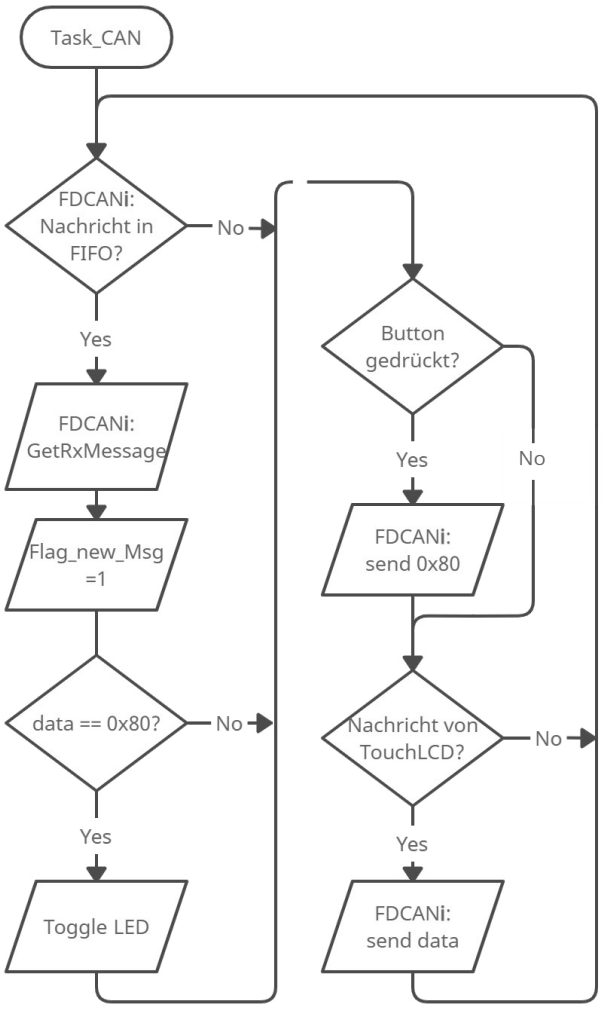


Abbildung 5.2.3: Flussdiagramm - Task CAN

5.2.3 Ansteuerung und Bedienung des Touchdisplays

Hier wird ein kurzer Überblick zur allgemeinen Grundfunktion des Displays gegeben. Die genaue Funktion bzw. der Ablauf der Programmsequenzen sowie Tasks wird in Kapitel 5 gegeben.

Initialisierung des Displays

Zu Beginn des Programmablaufes sollte die ID des verwendeten Displays ausgelesen werden. Dies funktioniert über die vordefinierte Funktion *readID()*, die die ID als Rückgabewert liefert. Anschließend kann mittels *tft_init(ID)* das Display initialisiert werden. Aus dem Datenblatt des ILI9486 Controllers, können die entsprechenden Kommandos und die damit verbundenen Datenbefehle entnommen werden. Als Beispiele hierfür sind das *Power Control Register*, das *Memory Control Register* oder auch die Gammawerte der Farben zu nennen. [30, 32]

Die voreingestellten Initialisierungsdaten von ControllersTech sind bereits ausreichend genug für den ILI9486.

Bildausgabe

Der Task *Display* übernimmt die Aufgabe der Bildausgabe und die Aktualisierung der Relaiskontakte im Display. Nach der eingangs erwähnten Initialisierung wird das Hintergrundbild (siehe Abb. 5.2.5) mithilfe der Funktion *drawRGBBitmap()* ausgegeben. Anschließend werden in einer Dauerschleife die jeweiligen Relaiszustände abgefragt und in Abhängigkeit davon die Bilder für einen geschlossenen oder offenen Schalter ausgegeben. Analog hierzu wird auch der Zustand (Ein/Aus) der Abschlusswiderstände abgefragt und ausgegeben.

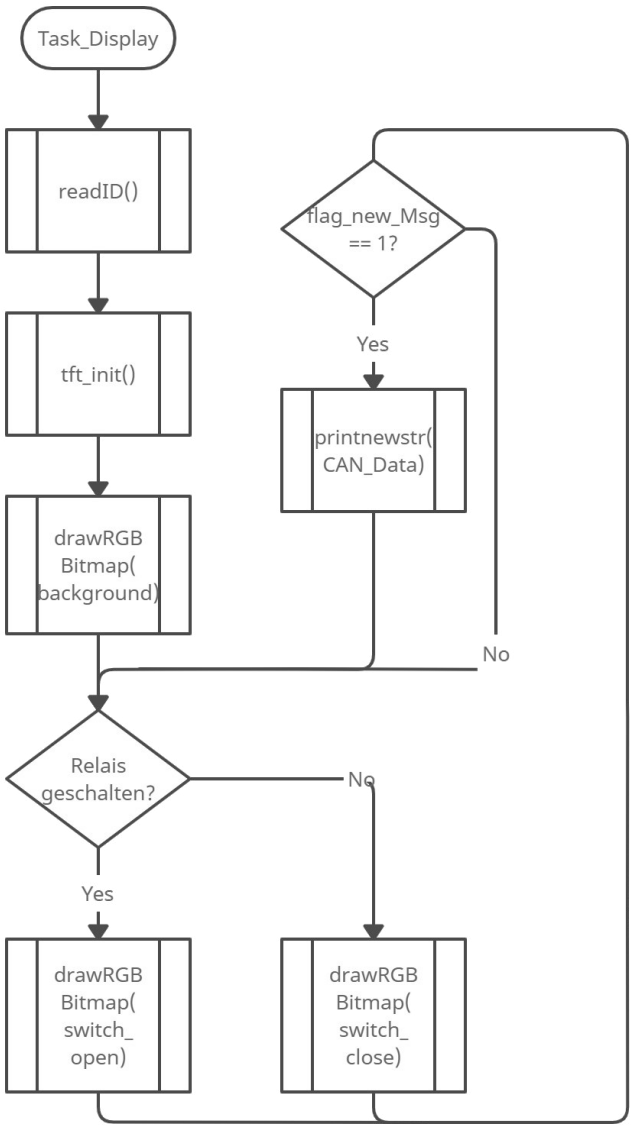


Abbildung 5.2.4: Flussdiagramm - Task Bildausgabe auf Display

Wenn eine neue CAN-Nachricht empfangen wurde (Flag für eine neue Nachricht ist 1), werden die zuletzt empfangenen Daten der Nachrichten für CAN1 und CAN2 auf dem Display mithilfe der *printnewstr()*-Funktion ausgegeben.

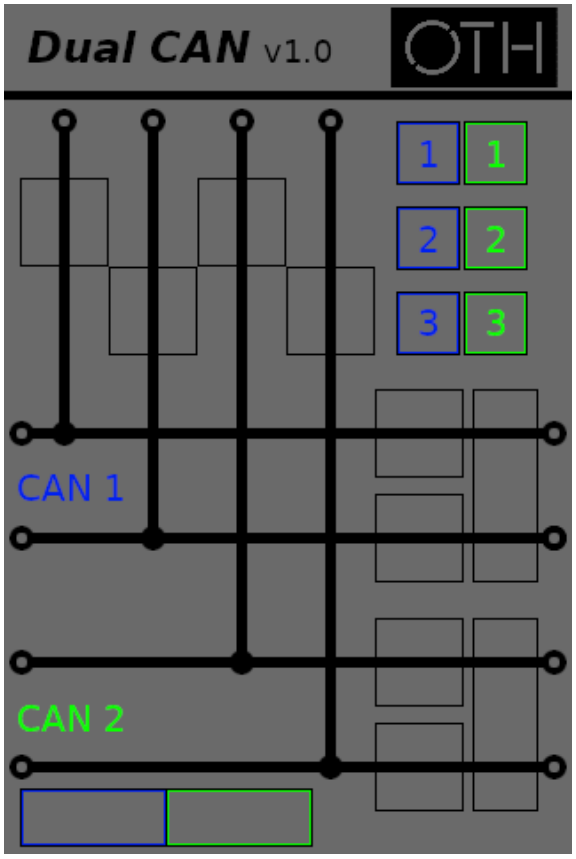


Abbildung 5.2.5: Displayanzeige - Hintergrundbild für weitere Anzeigen

Einlesen der Daten des Touchscreens

Um die Bedienung über das Touchdisplay zu ermöglichen, wird der Task *Touch* benötigt. Hier wird kontinuierlich der aktuelle Touchpoint eingelesen und in einem Struct mit 3 Werten (x-, y- und z-Koordinate) abgespeichert.
Wenn die z-Koordinate (Druckstärke) groß genug ist, wird die gemessene x- und y-Koordinate skaliert und weiterverarbeitet. Im Anschluss werden die aktuellen Koordinaten mit den definierten Schaltflächen verglichen. Bei einem passenden Bedienbefehl wird die zugehörige Aktion (Relais schalten oder Test-CAN-Nachrichten senden) ausgeführt. Abb. 5.2.6 zeigt das zugehörige Flussdiagramm zum beschriebenen Task.

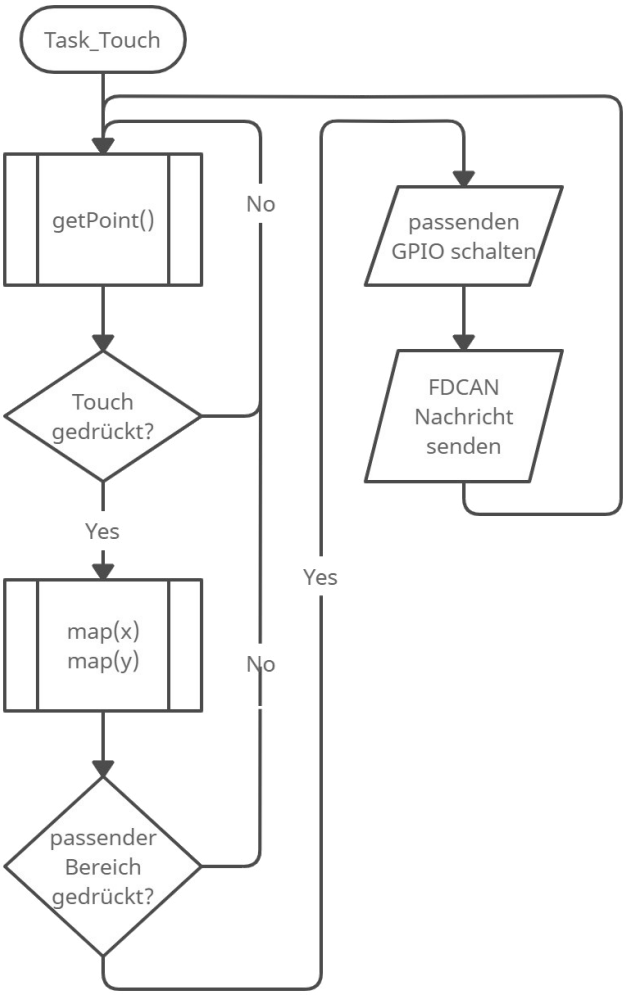


Abbildung 5.2.6: Flussdiagramm - Task Touchscreen Eingabe

Bedienung und GUI

Durch Tippen auf die angedeuteten Quadrate und Rechtecke in Abb. 5.2.5 kann des jeweilige Relais geschaltet werden. Diese wiederum schalten die Unterbrechungen bzw. aktivieren die Abschlusswiderstände. In Abb. 5.2.7 sind die aktivierten Unterbrechungen in CAN2 für das Low- und High-Signal zu erkennen.
Die Unterbrechungen je FIM sind zum einen schaltbar in der Stichleitung zum Microcontroller (stub) sowie in der Linienbusleitung selbst (bus line).
Die Test-CAN-Nachrichten können über die Zahlen 1, 2 und 3 im rechten, oberen Bereich gesendet werden. Im linken, unteren Bereich wird für jeden CAN-Bus die zuletzt empfangene Nachricht angezeigt.

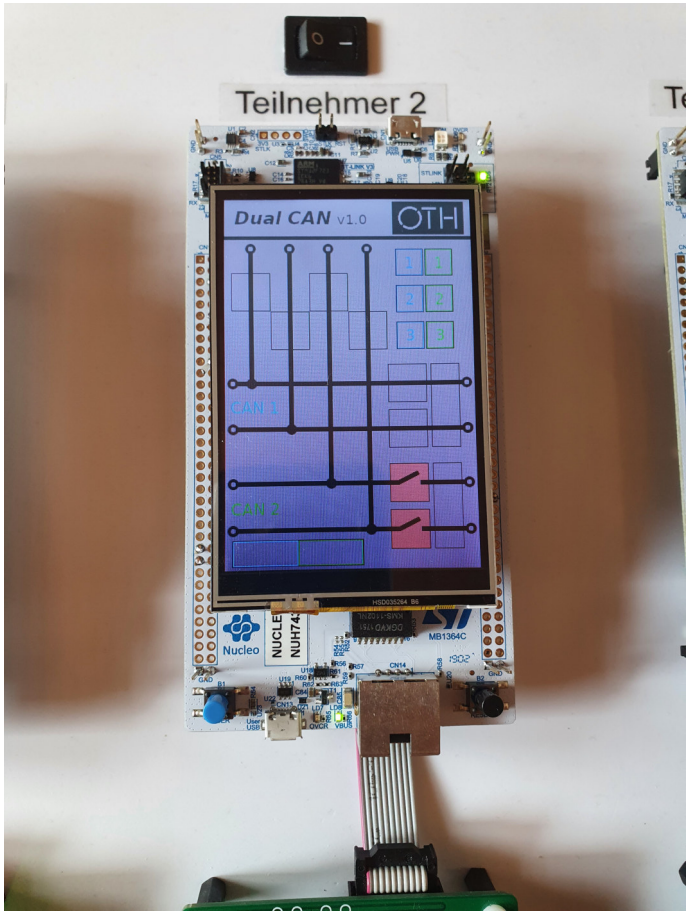
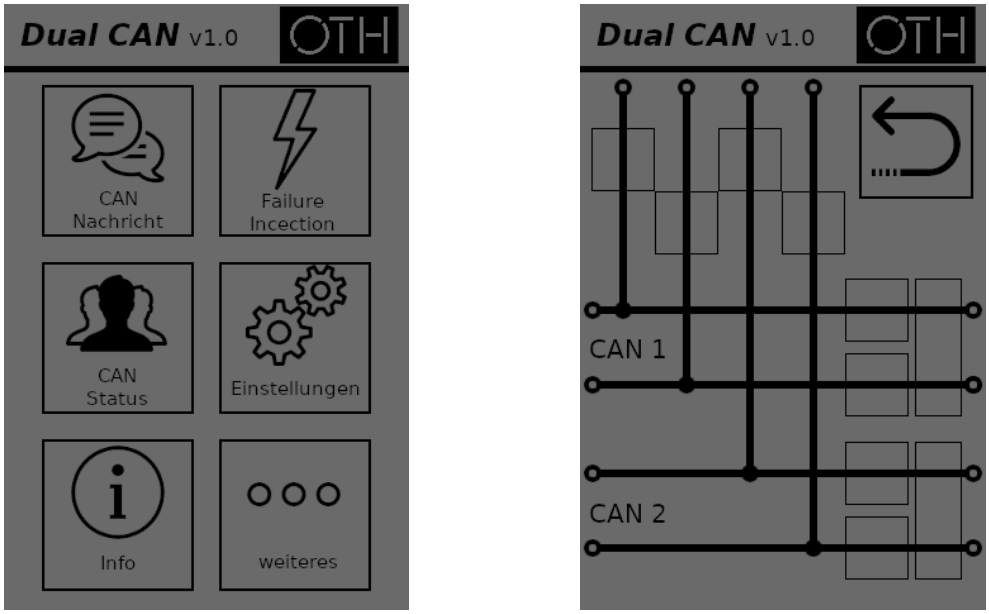


Abbildung 5.2.7: Displayanzeige - Unterbrechung in CAN2_L und CAN2_H (bus line)

Abb. 5.2.8 zeigt eine mögliche GUI als Erweiterung des Projekts um detailliertere Informationen auszugeben oder auch mehr Einstellungsmöglichkeiten des CAN-Netzwerks vorzunehmen. Um dies zu realisieren ist der SD-Kartenleser des Displays nötig. Der Programmspeicher des Microcontrollers reicht nur für ein Bild der gesamten Displaygröße aus.



(a) (b)

Abbildung 5.2.8: Anzeige auf dem Display: GUI Hauptmenü und Untermenü Failure Injection

6. Validierung und Fazit

Abschließend sollte die Funktionalität des Demonstratoraufbaus hinsichtlich des CAN-Netzwerks sowie der Bedienung über das FIB und auch des Touchdisplays untersucht werden. Nachfolgend wird ein kurzes Fazit gegeben.

6.1 Validierung

Zu Beginn wird die Funktion des Dual-CAN-Netzwerks getestet und die Ergebnisse präsentiert. Auch das Touchdisplay wird getestet und eventuelle Probleme bei der Bedienung aufgezeigt. In Abb. 6.1.1 ist der Demonstrator mit den angeschlossenen Geräten dargestellt.

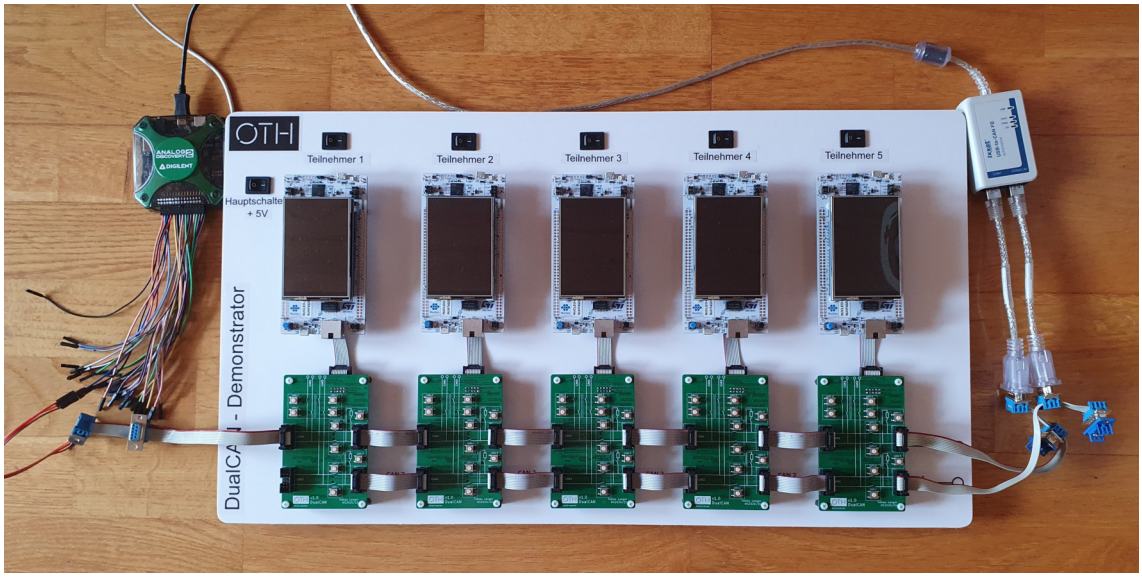


Abbildung 6.1.1: Testaufbau (Mitte) mit angeschlossenem AnalogDiscovery von Digilent (links) und Ixxat USB-to-CANFD (rechts)

Dual-CAN-Netzwerk und FIB

Es wird die Kommunikation der Microcontroller getestet und die Funktionalität des Dual-CAN-Netzwerks. Mithilfe des Digitaloszilloskops AnalogDiscovery von Digilent [36] können die CAN-Pegel genauer untersucht werden. Durch das Ixxat USB-to-CANFD [37] wird die gesamte Buslast aufgenommen und dargestellt. Die Nachrichten werden analysiert und mit der richtigen ID sowie den Daten der Nachricht wiedergegeben.

Im aufgespielten Programm werden testweise CAN-Nachrichten mit der ID 21 bei CAN1 bzw. 22 bei CAN2 gesendet. Die Nachrichten werden vom Ixxat korrekt aufgenommen und dargestellt. Es können ebenfalls Nachrichten vom Ixxat gesendet und vom Microcontroller empfangen werden (in diesem Fall mit der ID 20). In Abb. 6.1.2 ist ein Screenshot des Programms CanAnalyser Mini 3 mit einem Nachrichtenverlauf auf CAN1 dargestellt. Die eingestellte Geschwindigkeit des Busses ist 500/1000 kBit/s.

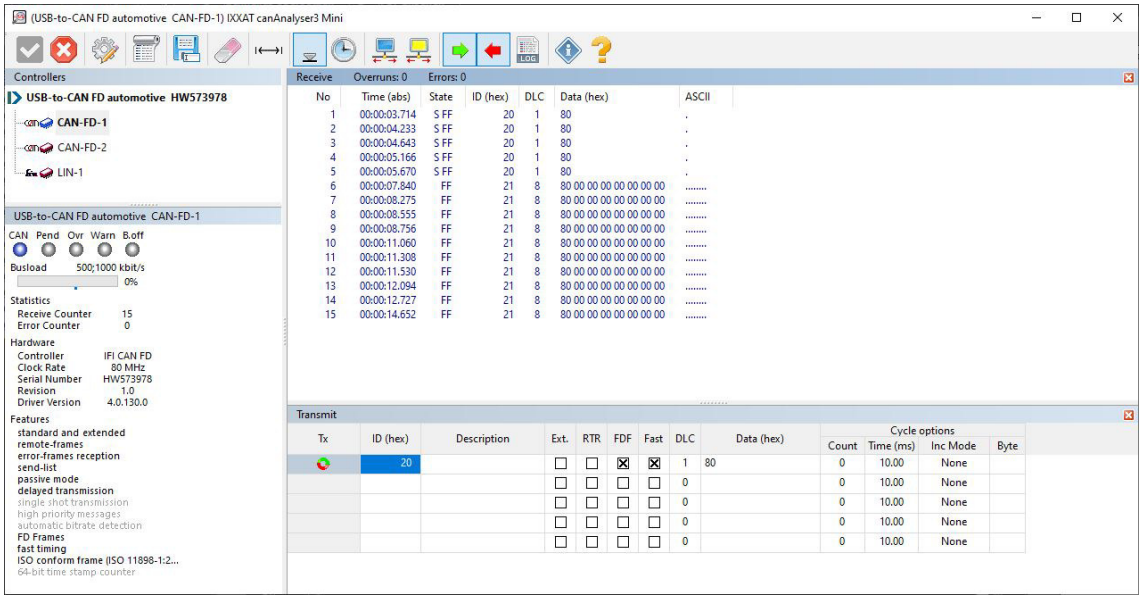


Abbildung 6.1.2: Screenshot der aufgenommenen CAN FD-Nachrichten des Ixxat (Programm: CanAnalyser Mini 3)

Es wurde noch keine Nachrichtenumleitung implementiert und dadurch nur der einfache Bus ohne weitere Fehlersimulation getestet. Nach Verbesserung der CAN-Wege auf dem FIB durch Fädeldrähte (siehe Abb. 6.1.3) schalten die Relais die korrekten Stellen im Bus.

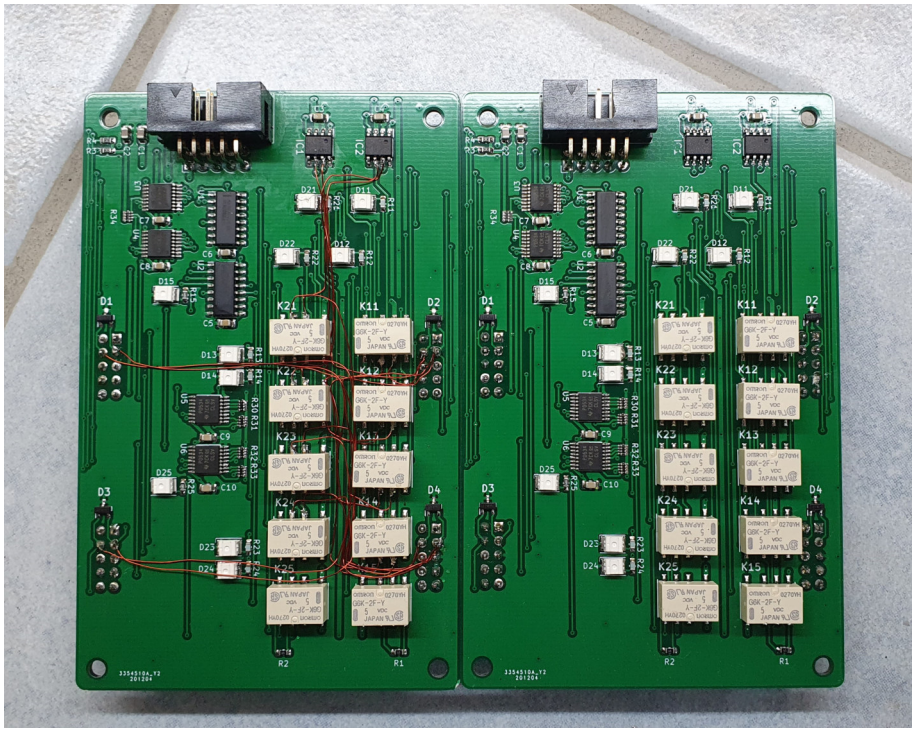


Abbildung 6.1.3: Verbesserung der Wege des CAN-Netzwerks mithilfe von Fädeldrähten

Außerdem wurde der CAN-Bus mithilfe des Digitaloszilloskops untersucht. Bei Abb. 6.1.4 ist deutlich die Notwendigkeit der Abschlusswiderstände im Netzwerk zu erkennen. Aufgrund von Reflexionen in den Leitungen ist keine eindeutige CAN-Nachricht mehr zu erkennen. Abb. 6.1.5 zeigt im Gegensatz hierzu die korrekt übermittelte Nachricht, bei der sich auch das Umschalten der Bitrate im Datenteil der CAN FD-Nachricht sowie das ACK-Bit (die Bestätigung der anderen Teilnehmer für die Plausibilität der Nachricht) erkennen lässt.

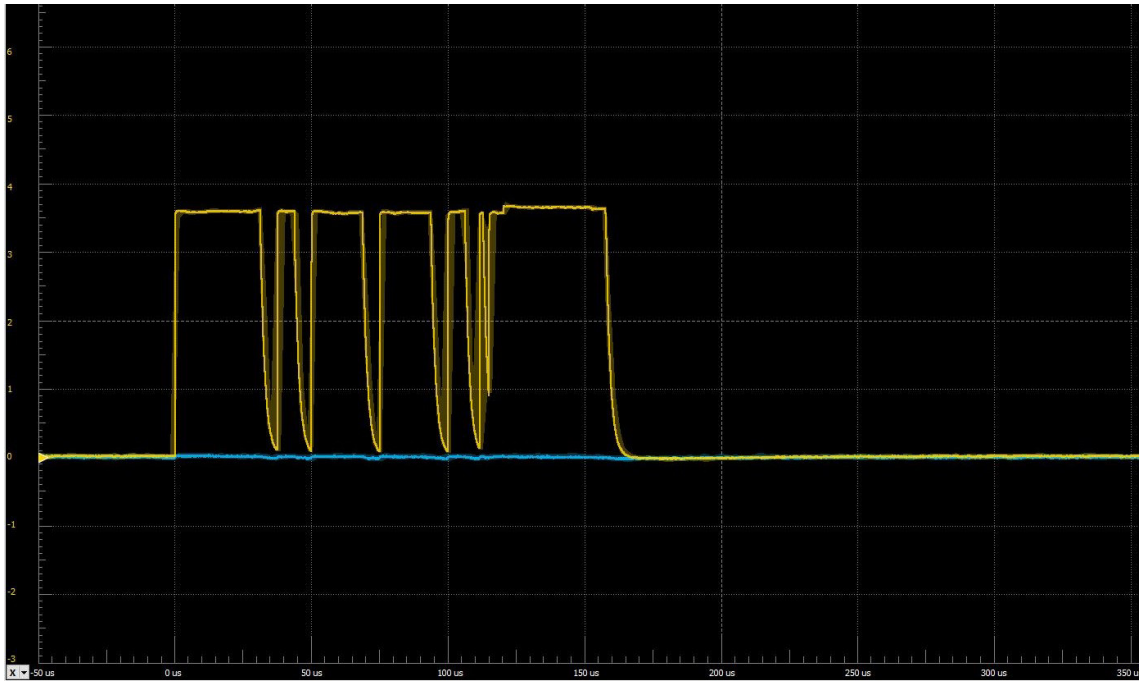


Abbildung 6.1.4: Screenshot AnalogDiscovery: Messung des CAN1 ohne Abschlusswiderstand

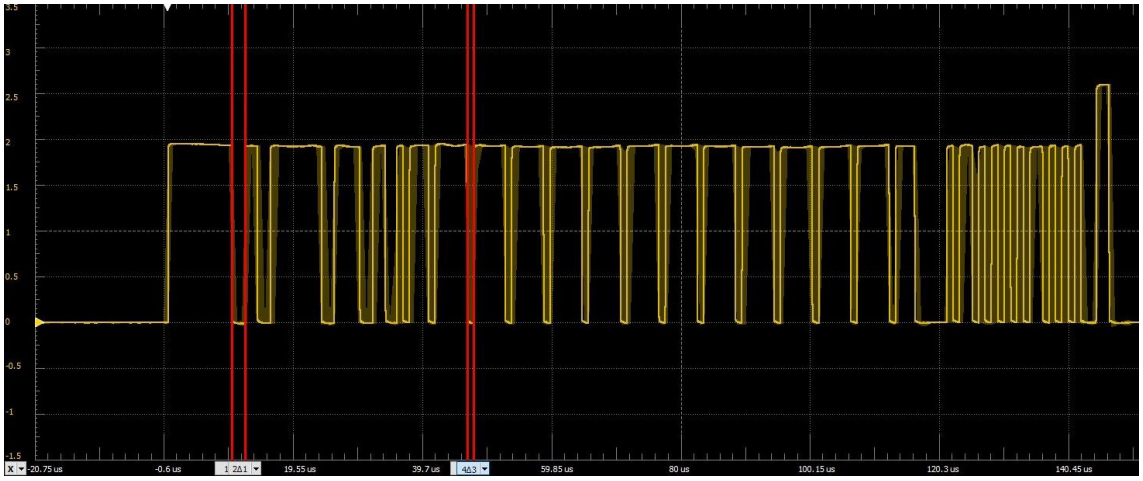


Abbildung 6.1.5: Screenshot AnalogDiscovery: Messung des CAN1 mit Abschlusswiderstand und Bit-Rate-Switching (CAN FD)

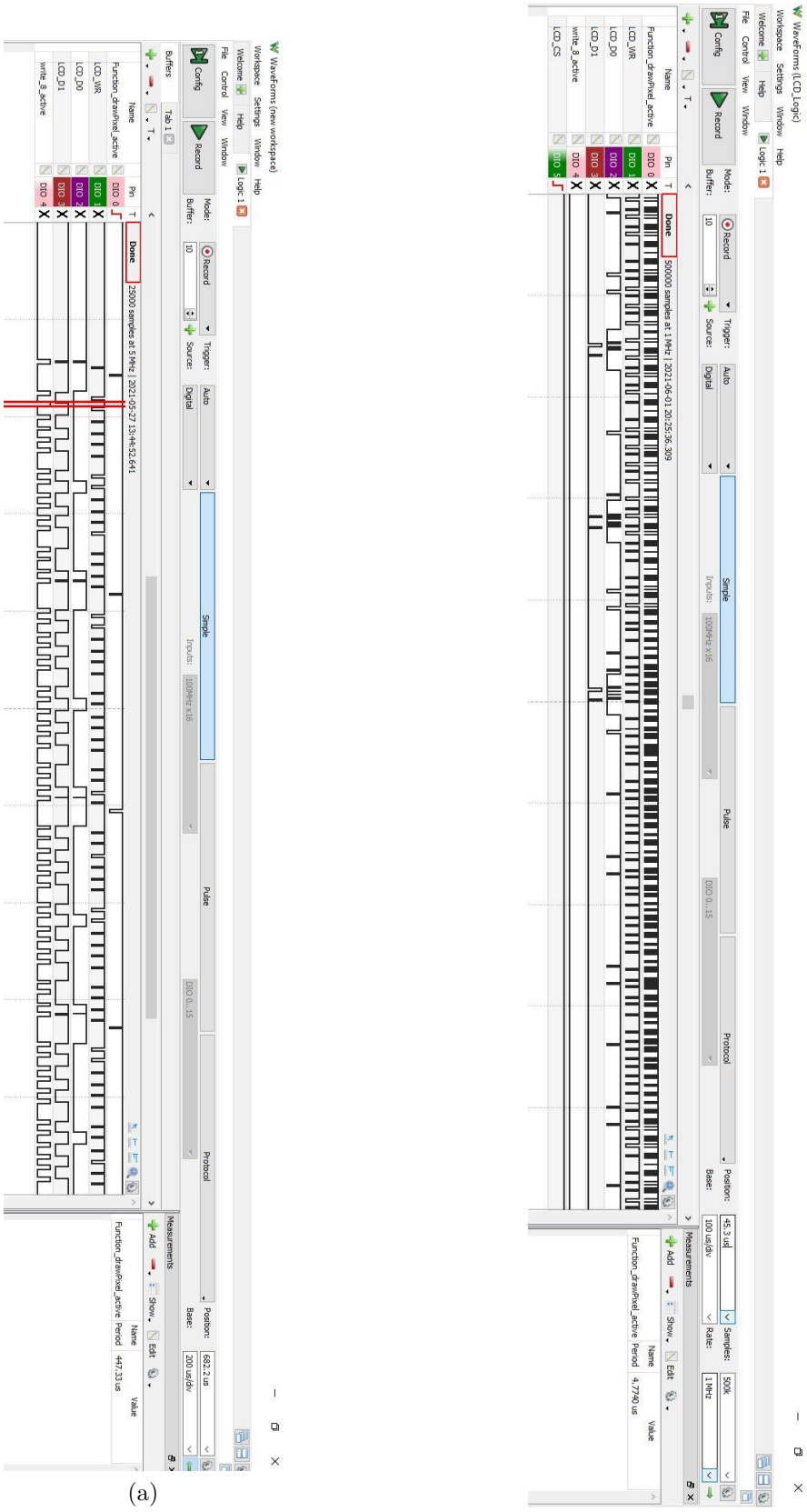
Touchdisplay

Die Übertragung der Daten auf das Display wurde ebenfalls mithilfe des Analog Discovery analysiert. Mit der Bibliothek von ControllersTech war die Übertragung von Bitmap-Dateien an das Display sehr langsam und träge (siehe Abb. 6.1.6a). Es wurde mithilfe eines GPIO-Pins die Dauer der *drawPixel()*-Funktion aufgezeichnet und ausgegeben. Im ersten Fall (ohne jegliche Optimierung) dauerte die Übertragung eines Pixels ca. 450 μs. Diese Zeit ist sehr langsam für ein Display mit 480x320 Pixeln (ca. 70 Sekunden/Bild).
Durch Optimierungen, wie die neue *drawRGBBitmap()*-Funktion, konnte eine deutlich bessere Geschwindigkeit von ca. 4,8 μs pro Pixel erreicht werden (ca. 1 Sekunde/Bild).

Die Bedienung des Touchdisplay gestaltet sich als etwas schwieriger. Die grundlegende Funktion wurde implementiert, dennoch müsste das Programm auf das jeweilig aufgesetzte Display und die gelieferten Spannungswerte kalibriert werden.
Getestet wurde die Touchfunktion mit einer Anzeige, die durch die Berührung hervorgerufenen Werte direkt als Pixel (oder auch etwas größeren Punkt) wieder anzeigt. So kann die Funktionalität gezielt getestet werden.

6.2 Fazit

Der Demonstratoraufbau für Dual-CAN funktioniert in seiner grundlegenden Funktion bereits sehr gut und wartet auf weitere Erweiterungen bzw. noch anstehende Verbesserungen. Man kann CAN-Nachrichten zwischen den einzelnen Teilnehmern versenden und auch empfangen. Die Analyse des CAN-Netzwerks erfolgte durch das Digitaloszilloskop AnalogDiscovery sowie mithilfe des Ixxat USB-to-CANFD. Die Fehlerinjektion funktioniert einwandfrei und kann gezielt durch die vorhandenen Bedieneinheiten Drahtbrüche simulieren. Diese Fehlerinjektion kann manuell durch Betätigung der Taster oder mittels Touchdisplay erfolgen. Außerdem sind die Fehlerinjektionen auch durch ein Programm auf dem Microcontroller schaltbar. Beim Touchdisplay besteht noch geringer Verbesserungsbedarf, jedoch reicht die Funktionalität für die Anwendung des Demonstrators aus. Weitere Details hierzu folgen im nächsten Kapitel.



(a) (b)

Abbildung 6.1.6: Screenshot AnalogDiscovery: Analyse der Datenübertragung zum Display, Übertragungsdauer für ein Pixel, rechts: nicht optimiert, links: optimiert

7. Ausblick

Dieses Kapitel enthält mögliche Verbesserungsansätze des Demonstratoraufbaus, des Touchdisplays und des FIBs. Außerdem werden die Erweiterungen, die im nächsten Schritt erfolgen, erläutert.

7.1 Verbesserungen

Die Funktionalität des Demonstratoraufbaus ist für die derzeitige Anwendung ausreichend, jedoch noch nicht optimal. Im Folgenden werden die Verbesserungen aufgezählt:

- Die Bildwiederholrate des Displays ist für diese Anwendung ausreichend und es wird das Wichtigste schnell angezeigt. Dennoch werden Bilder, die das ganze Display ausfüllen, nur etwa in einem Zyklus von einer Sekunde erneuert.
  Da bereits die CPU-Clock auf die maximale Frequenz eingestellt ist, könnte man die Priorisierung der Tasks ändern, um eine flüssigere Bildwiederholung zu erreichen. Eventuell gibt es auch im Programmcode selbst, also der Displayansteuerung und -datenübertragung Verbesserungspotential.
  Da das Nucleo STM32H7 Board bereits eine LCD-TFT-Schnittstelle für Displays bis zu einer XGA-Auflösung (1024x768 Pixel) bereitstellt, wäre es eine Überlegung wert, darauf umzusteigen. Hierzu wird aber eine andere Hardware (Display) benötigt.

- Als weitere Verbesserung bei der Displaydatenübertragung bietet sich außerdem eine Änderung der Belegung der GPIO-Pins an. Wie in Tabelle 4.1.1 zu sehen ist, werden viele verschiedene Ports verwendet. Dies macht die Datenübertragung in der Hinsicht langsamer, dass immer mehrere Register angesprochen und von Input auf Output geändert werden (und das bei jedem Pixel). Wenn beispielsweise nur GPIO_E verwendet werden würde, könnte man ein paar Befehle einsparen und die Bildwiederholrate erneut verbessern.

- Wenn der Gedanke naheliegt, die verwendeten Pins auf dem Nucleo STM32H7 zu verändern, muss eine Zwischenplatine entwickelt werden, die auf das Board

aufgesteckt wird und gleichzeitig die passende Aufnahme für das Display besitzt.
Dies kann dazu genutzt werden eine zweite Version des FIBs zu designen. Das FIB könnte auch als Aufsteckplatine entwickelt werden und die Bedienung sowie Anzeige könnte rein über das Touchdisplay erfolgen.

- Im gleichen Zug muss bei der Entwicklung der zweiten Version des FIBs die Verdrahtung der Relais im CAN-Netzwerk verbessert werden. Diese wurden nachträglich mithilfe von Fädeldrähten angepasst und ermöglichen so übergangsweise die Funktion (siehe Abb. 6.1.3).

- Um der Bedienung alleine über das Touchdisplay zu vertrauen, muss die Touchfunktion selbst ebenfalls optimiert werden. Hilfreich wäre hier eine Kalibrierungsfunktion, welche spezifisch auf das jeweilige Display abgestimmt ist. Ob die Kalibrierung bei jedem Start erforderlich wäre oder ob es ausreichen würde die Daten einmal zu speichern und wieder abzurufen muss genauer untersucht werden.

7.2 Erweiterungen

Es bieten sich viele Erweiterungsmöglichkeiten an, die im weiteren Verlauf der Arbeit in Betracht gezogen werden sollten. Hier werden einige Beispiele aufgeführt:

- Wie in Kapitel 5.2.3 erwähnt, ist eine mögliche Erweiterung eine GUI zu zeichnen bzw. zu programmieren. Dies erleichtert die Bedienung enorm und gibt so viele Informationen an den Benutzer weiter. Es können auch andere weitere Funktionen hier untergebracht werden, wie zum Beispiel der Start der Kalibrierungsfunktion, die Failure Injection selbst oder die Anzeige des aktuellen Status des CAN-Netzwerks.

- Mithilfe der GUI kann wie zuvor erwähnt eine detailreiche Anzeige über den aktuellen Status des CAN-Netzwerks realisiert werden. Hierzu zählen Informationen zu den einzelnen Teilnehmern im Netzwerk (beispielsweise mittels Heartbeat-Nachrichten) oder auch die aktuellen Drahtbrüche im Netzwerk.

- Als wichtigsten Punkt muss natürlich der Demonstratoraufbau die Nachrichtenumleitung des Dual-CAN beherrschen und melden können. Nur so kann die Funktion des Dual-CAN getestet und analysiert werden.

- Im gleichen Zug kann hier der Dual-CAN auch analysiert werden. Hier sollten die Kategorien wie Funktionalität des Dual-CAN und die Ausfallsicherheit genauer beobachtet werden. Wie im Kapitel 2.2 bereits erwähnt, sollten auch die theoretischen Aspekte bei der Analyse nicht fehlen.

Literaturverzeichnis

[1] Jose Rufino. *Dual-media redundancy mechanisms for CAN*. 1997. URL: http://dario.di.fc.ul.pt/downloads/cstc-rt-9701.pdf.

[2] Vector. *Vector E-Learning - CAN*. 12.06.2021. URL: https://elearning.vector.com/.

[3] C. Guerrero, G. Rodriguez-Navas und J. Proenza. „Design and implementation of a redundancy manager for triple redundant CAN controllers“. In: *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*. 2002, 2294–2299 vol.3. DOI: 10.1109/IECON.2002.1185330.

[4] A. Reindl u. a. *Comparative Analysis of CAN, CAN FD and Ethernet for Networked Control Systems*. Hrsg. von C,A newsletter. 2021.

[5] F. l. Ni u. a. „Joint Fault-Tolerant Design of the Chinese Space Robotic Arm“. In: *2006 IEEE International Conference on Information Acquisition*. 2006, S. 528–533. DOI: 10.1109/ICIA.2006.305789.

[6] C. E. Lin und H. Yen. „A Prototype Dual Can-Bus Avionics System for Small Aircraft Transportation System“. In: *2006 ieee/aiaa 25TH Digital Avionics Systems Conference*. 2006, S. 1–10. ISBN: 2155-7209. DOI: 10.1109/DASC.2006.313766.

[7] T. Langer. „Dual-CAN (FD): Betrachtung hinsichtlich Fehlerdetektion, Nachrichtenumleitung und funktionaler Sicherheit“. FMS Bericht WS2020/21. Regensburg: OTH Regensburg, 2021. DOI: 10.35096/OTHR/PUB-1915.

[8] H. Xiang-Dong, Y. Hui-Mei und Z. Xiao-Xu. „Design of dual redundancy CAN-bus controller based on FPGA“. In: *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*. 2013, S. 843–847. ISBN: 2158-2297. DOI: 10.1109/ICIEA.2013.6566484.

[9] X. Zhai u. a. „Design of Double Redundant Interface Based on CAN Bus for Nodes of FCS“. In: *2010 Second WRI Global Congress on Intelligent Systems*. 2010, S. 394–396. ISBN: 2155-6091. DOI: 10.1109/GCIS.2010.187.

[10] F. G. R. Neves und O. Saotome. „Comparison between Redundancy Techniques for Real Time Applications“. In: *Fifth International Conference on Information Technology: New Generations (itng 2008)*. 2008, S. 1299–1300. DOI: 10.1109/ITNG.2008.229.

[11] Jun Yang u. a. „Redundant design of A CAN BUS Testing and Communication System for space robot arm“. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. 2008, S. 1894–1898. DOI: 10.1109/ICARCV.2008.4795817.

[12] H. Cao u. a. „Marine Main Engine Remote Control System with redundancy CAN bus based on distributed processing technology“. In: *2010 International Conference on Intelligent Control and Information Processing*. 2010, S. 638–640. DOI: 10.1109/ICICIP.2010.5564343.

[13] D. Shea u. a. *Prototype Development of the SQX-1 Autonomous Underwater Vehicle*. St. John's, NL, Canada, 2009. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5278141.

[14] STMICROELECTRONICS. „STM32H742, STM32H743/753 and STM32H750 Value line advanced Arm®-based 32-bit MCUs - Reference manual“. In: (). URL: https://www.st.com/resource/en/reference_manual/dm00314099-stm32h742-stm32h743-753-and-stm32h750-value-line-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf.

[15] arm mbed. *NUCLEO-H743ZI2*. URL: https://os.mbed.com/platforms/ST-Nucleo-H743ZI2/.

[16] Microchip Technology Incorporated. *MCP2561/2FD High-Speed CAN Flexible Data Rate Transceiver*. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/20005284A.pdf.

[17] O. N. Semiconductor. *NUP2105L - 27 V ESD Protection Diode, Dual Line CAN Bus Protector*. URL: https://www.onsemi.com/pdf/datasheet/nup2105l-d.pdf.

[18] V. Schneider. „Communication in multi-microcontroller systems“. FMS Bericht WS2020/21. Regensburg: OTH Regensburg, 2021. DOI: 10.35096/OTHR/PUB-1915.

[19] Reinhard Rahner. *I2C Bus - Adressierung - Datenübertragung - Zeitdiagramm - Arduino UNO - Ünterricht - Lernmaterial - Mikrocontroller - serielle Kommunikation - MINT"*. 18.06.2021. URL: https://www.rahner-edu.de/grundlagen/signale-richtig-verstehen/i2c-besser-verstehen-1/.

[20] Texas Instruments, Incorporated [SCPS124 und H]. *PCA9534 Remote 8-Bit I2C and SMBus Low-Power I/O Expander With Interrupt Output and Configuration Registers datasheet (Rev. H)*. URL: https://www.ti.com/lit/ds/symlink/pca9534.pdf?&ts=1589463558594.

[21] OMRON. *G6K*. URL: https://cdn-reichelt.de/documents/datenblatt/C300/G6K%23OMR.pdf.

[22] STMICROELECTRONICS. *ULN200xx.fm*. URL: https://www.st.com/resource/en/datasheet/uln2001.pdf.

[23] OMRON. *B3S*. URL: https://cdn-reichelt.de/documents/datenblatt/C200/630425.pdf.

[24] Agilent. *HSMx-A10x-xxxxx PLCC-2 Surface Mount LED Indicator: Data Sheet*. URL: http://partkeepr.othr.de/data/media/HSMx-A10x-xxxxx.pdf.

[25] *LED-Vorwiderstandsrechner*. 12.06.2021. URL: https://www.elektronik-kompendium.de/sites/bau/1109111.htm.

[26] LCD Wiki. *3.5inch Arduino 8BIT Module MAR3501 User Manual*. URL: http://www.lcdwiki.com/res/MAR3501/3.5inch_Arduino_8BIT_Module_MAR3501_User_Manual_EN.pdf.

[27] LCD Wiki. *3.5inch Arduino Display-UNO – LCD wiki*. 16.12.2019. URL: http://www.lcdwiki.com/3.5inch_Arduino_Display-UNO.

[28] GitHub. *adafruit/Adafruit-GFX-Library*. 9.06.2021. URL: https://github.com/adafruit/Adafruit-GFX-Library.

[29] GitHub. *prenticedavid/MCUFRIEND_kbv*. 9.06.2021. URL: https://github.com/prenticedavid/MCUFRIEND_kbv.

[30] ControllersTech. *Interface TFT display with STM32 » ControllersTech*. 2019. URL: https://controllerstech.com/interface-tft-display-with-stm32/.

[31] ControllersTech. *Microsecond/Nanosecond delay in STM32 » ControllersTech*. 2017. URL: https://controllerstech.com/create-1-microsecond-delay-stm32/.

[32] hyliao. *ILI9486_SPEC_V001*. URL: http://www.lcdwiki.com/res/MAR3501/datasheet_ILI9486.pdf.

[33] NewImageUser. *"How can I convert between RGB565 and RGB24 image formats in MATLAB?"* 2010. URL: https://stackoverflow.com/questions/3578265/how-can-i-convert-between-rgb565-and-rgb24-image-formats-in-matlab.

[34] Gesas. „Resistive vs. kapazitive Touchscreens: Vor- & Nachteile“. In: *gesas.de* (11.11.2020). URL: https://www.gesas.de/de/news/resistive-touchscreens-vs-kapazitive-touchscreens.html.

[35] GitHub. *adafruit/Adafruit_TouchScreen*. 24.06.2021. URL: https://github.com/adafruit/Adafruit_TouchScreen.

[36] Digilent. *Analog Discovery 2 Reference Manual - Digilent Reference*. 21.07.2021. URL: https://reference.digilentinc.com/test-and-measurement/analog-discovery-2/reference-manual.

[37] Ixxat. „USB to CAN FD Handbuch Deutsch“. In: (2019). URL: https://cdn.hms-networks.com/docs/librariesprovider8/ixxat-english-new/pc-can-interfaces/manuals-design-guides/usb-to-can-fd-handbuch-deutsch.pdf?sfvrsn=f5ce4bd7_10.

Anhang

Inhalt des beigefügten Datenträgers:

- Präsentation als .pdf
- Schaltplan und Platinendesign der FIB-Platine als KiCad-Projekt
- Programmcode des Nucleo STM32H7 Boards als STM32CubeIDE-Projekt
- Programm zur Umwandlung von Bitmaps in 565-codierte Arrays in Textform als .exe
- sonstige verwendete Bilder und Datenblätter