Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Autonome Quartiersmobilität

MIKROCONTROLLERANSTEUERUNG ÜBER PLECS

Wintersemester 2021/2022

Betreuer:

Prof. Dr. Rudolph
M. Sc. Michael Brüns

Teilnehmer:

Ahmed Waleed Abdulhak Noman	2382555
Van Uoc Phung	2436401
Anh Quang Nguyen	2422165

Hamburg, den 31.03.2022

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Aufgabenbeschreibung

Ein bereitgestelltes System, das aus einem Elektromotor und einem Mikrocontroller (M1) besteht, sollte jetzt von einem anderen Mikrocontroller (M2) gesteuert werden. Die Steuerungsdaten des Elektromotors werden von Mikrocontroller (M2) an Mikrocontroller (M1) übertragen. Danach kann der Mikrocontroller (M1) den Elektromotor steuern.

Zur Übertragung der Daten wird CAN-Bus verwendet. In diesem Projekt sollte ein CAN-Bus Protokoll entwickelt werden, um die Kommunikation zwischen beiden Mikrocontroller herzustellen und den Elektromotor steuern zu können.

1.2. Hardware

In diesem Projekt werden die folgenden Hardware verwendet:

- Texas Instruments C2000 LaunchPad XL (F28379D)

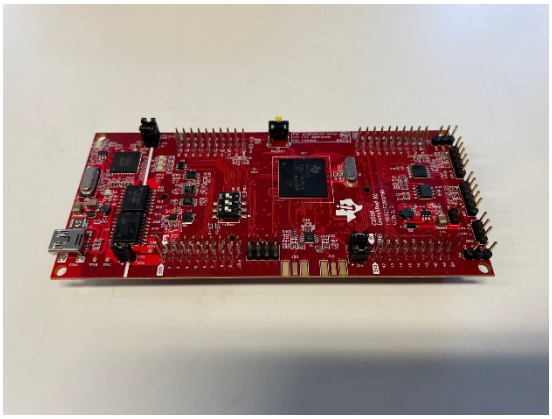

Abbildung 1: C2000 LaunchPad XL F28379D

- Motor Driver Booster Packs BOOSTXL-DRV8305EVM

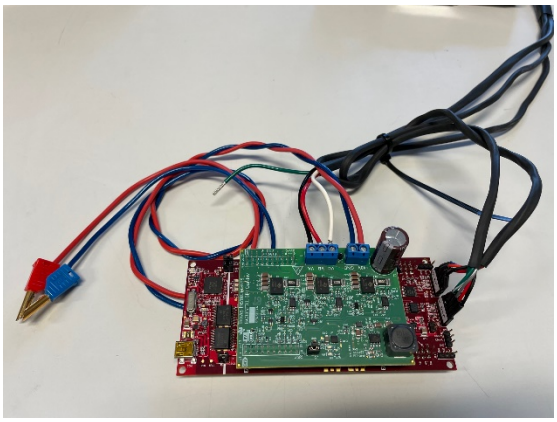

Abbildung 2: BOOSTXL-DRV8305EVM

- Elektromotor 2MTR-DYNO

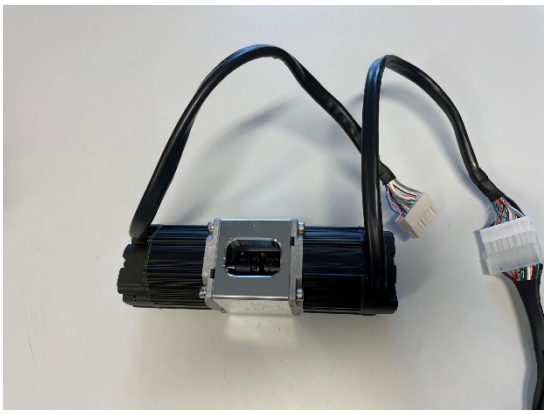

Abbildung 3: Elektromotor 2MTR-DYNO

- Jumper Wire Kabel DEBO Entwicklerboards

Abbildung 4: Jumper Wire Kabel

- Spannungsquelle VOLTCRAFT DPPS-60-15

Abbildung 5: Spannungsquelle VOLTCRAFT DPPS-60-15

- Verbindungskabel

Abbildung 6: Multimeter Kabel

1.3. Software

Um die Mikrocontroller von Texas Instruments zu programmieren, kann die Software CodeComposerStudio verwendet werden. Im Rahmen dieses Projekts wird ersatzweise PLECS verwendet.

PLECS ist ein Software-Tool für Simulationen elektrischer Schaltkreise auf Systemebene, das von der Firma Plexim entwickelt wurde. Es wurde speziell für die Leistungselektronik entwickelt, kann jedoch für jedes elektrische Netzwerk verwendet werden. PLECS verfügt über eine PLECS Coder-Funktion für die Implementierung. Mit dem separat erhältlichen PLECS Coder kann C-Code für eine mit dem PLECS Blockset modellierte Schaltung erzeugt werden.

1.4. Zielsetzung

Das Ziel ist es, die Kommunikation CAN-Bus zwischen zwei Mikrocontroller aufzubauen. Daneben kann man die Hardwarearchitektur darstellen. Die Verbesserung der Lernfähigkeit der Software und Hardware sollte ein weiteres Ziel gesetzt werden.

2. Theorienanwendung

2.1. CAN-Bus

Der CAN-Bus (Controller Area Network) ist ein serielles Bussystem und gehört zu den Feldbussen. Es wurden 1983 vom Unternehmen Bosch für die Fahrzeugtechnik entwickelt. Mittlerweile ist CAN aber auch in der Industrieautomation und anderen Bereichen (Gebäudeautomatisierung, Industriemaschinen, ...) zum gängigen Standard geworden.

2.2. Vorteile von CAN-Bus:

- Verkabelung einzelner Systeme, Sensoren und Aktoren reduzieren

- Zugriffskonflikte werden erkannt und aufgelöst
- Wichtige Nachrichten haben Vorrang und werden nicht verworfen
- Serieller Feldbus zur Übertragung kleiner Datenmenge
- Multimaster-Fähigkeit

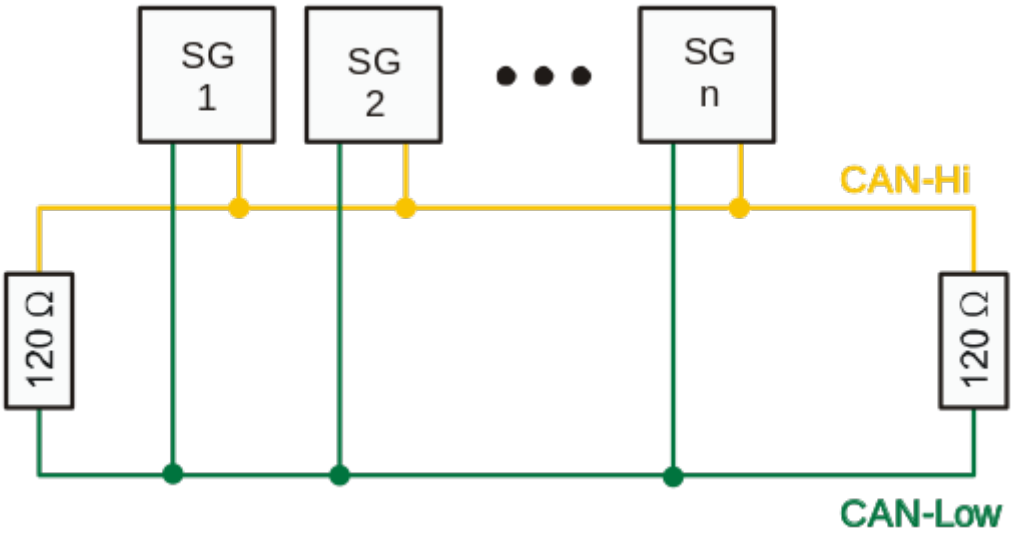

Abbildung 7: Linearer CAN-Bus (SG: Steuergerät)

Quelle: File:CAN-Bus Elektrische Zweidrahtleitung.svg - https://en.wikipedia.org

2.3. Physikalische Eigenschaften

- 2 Leitungen: CAN-High und CAN-Low
- Netzknoten: In einem CAN-System kommunizieren Netzknoten (nodes) miteinander.
- Topologie: Bus- oder Linienstruktur
- Leistungsabschluss: 120 Ω an jedem Leitungsende
- Übertragungsmedium: Zweidrahtleitung verdrillt und abgeschirmt

2.4. Funktionsprinzip

2.4.1. Buszugriff

Der Buszugriff (Arbitrierung) erfolgt nach dem CSMA/CA-Verfahren (Carrier Sense Multiple Access with Collision Avoidance). Die Übertragung einer Nachricht beginnt mit dem SOF-Bit (Start of Frame) und der nachfolgende Identifier enthält die Kennung und somit die Priorität der Nachricht. Die von irgendeinem Knoten auf dem CAN-Bus gesendete Datennachricht enthält nicht die Adresse des sendenden Knotens oder eines erwarteten empfangenden Knotens. Stattdessen wird der Inhalt der Nachricht mit einem Identifier (ID) gekennzeichnet, die im gesamten Netzwerk eindeutig ist. Alle anderen Knoten im Netzwerk empfangen die Nachricht, und jeder

Knoten führt eine Akzeptanzprüfung der ID durch, um festzustellen, ob die Nachricht für diesen Knoten relevant ist. Wenn die Nachricht relevant ist, wird sie verarbeitet; andernfalls wird es ignoriert.

2.4.2. Buspegel

Die Signalübertragung erfolgt über zwei Leiter asynchron nach symmetrischer Übertragung. Der CAN-Bustransceiver wertet Spannungsdifferenzen zwischen CAN-High und CAN-Low aus. Im rezessiven Zustand (logisch 1) sind beide Ausgangsstufen hochohmig. Im dominanten Zustand (logisch 0) wird CAN-High über den Bustreiber an Vcc und CAN-Low an GND gelegt.

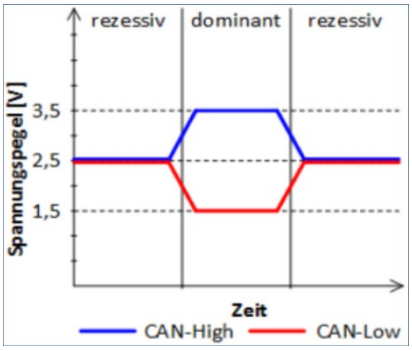

Abbildung 8: Spannungspegel

2.4.3. Leitungscodierung

CAN-Bus benutzt die Non Return to Zero (NRZ)-Codierung mit Bit-Stuffing nach 5 aufeinander folgenden gleichen Bits.

2.4.4. Daten-Frame-Struktur

In diesem Projekt verwenden wir CAN 2.0B, dessen Framework folgend gezeigt wird.

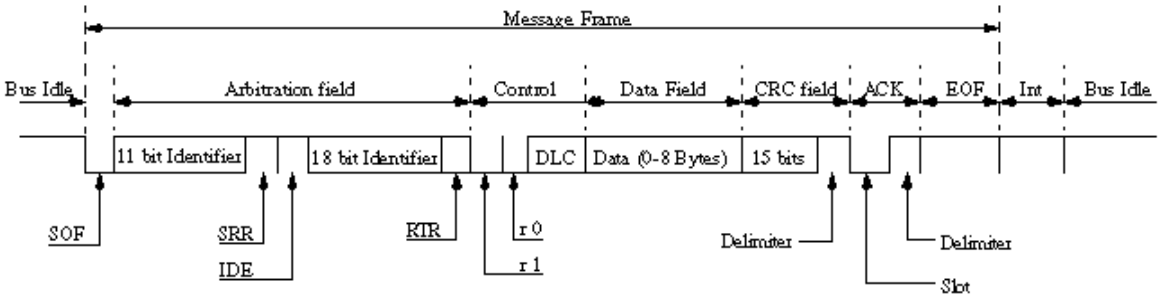

Abbildung 9: CAN 2.0B-Daten-Frame

- SOF (Start Of Frame)
- Identifier: 29 Bits bei CAN 2.0B
- RTR-Bit (Remote Transmission Request): logisch 0 für Daten-Frame
- Kontrollfeld (Control) = 6 Bit
  - Identifier Extension (IDE) = 1 Bit
  - Reserved = 1 Bit
  - Data Length Code (DLC) = 4 Bit (Anzahl der Bytes im Datenfeld, 0 bis 8 Bytes, Werte 9 bis 15 werden nicht unterstützt)
- Datenfeld (Data): 0 bis 8 Byte
- Prüfsummenfeld (CRC) = 15 Bit
- Bestätigungsfeld (ACK) = 2 Bit

- End of Frame (EOF) = 7 Bit (rezessiv)
- Intermission (IFS – Intermission Frame Space) = 3 Bit (= min. Anzahl der Bits, die aufeinanderfolgende Botschaften trennt)

3. Systemkonzept

3.1. Blockschaltbild

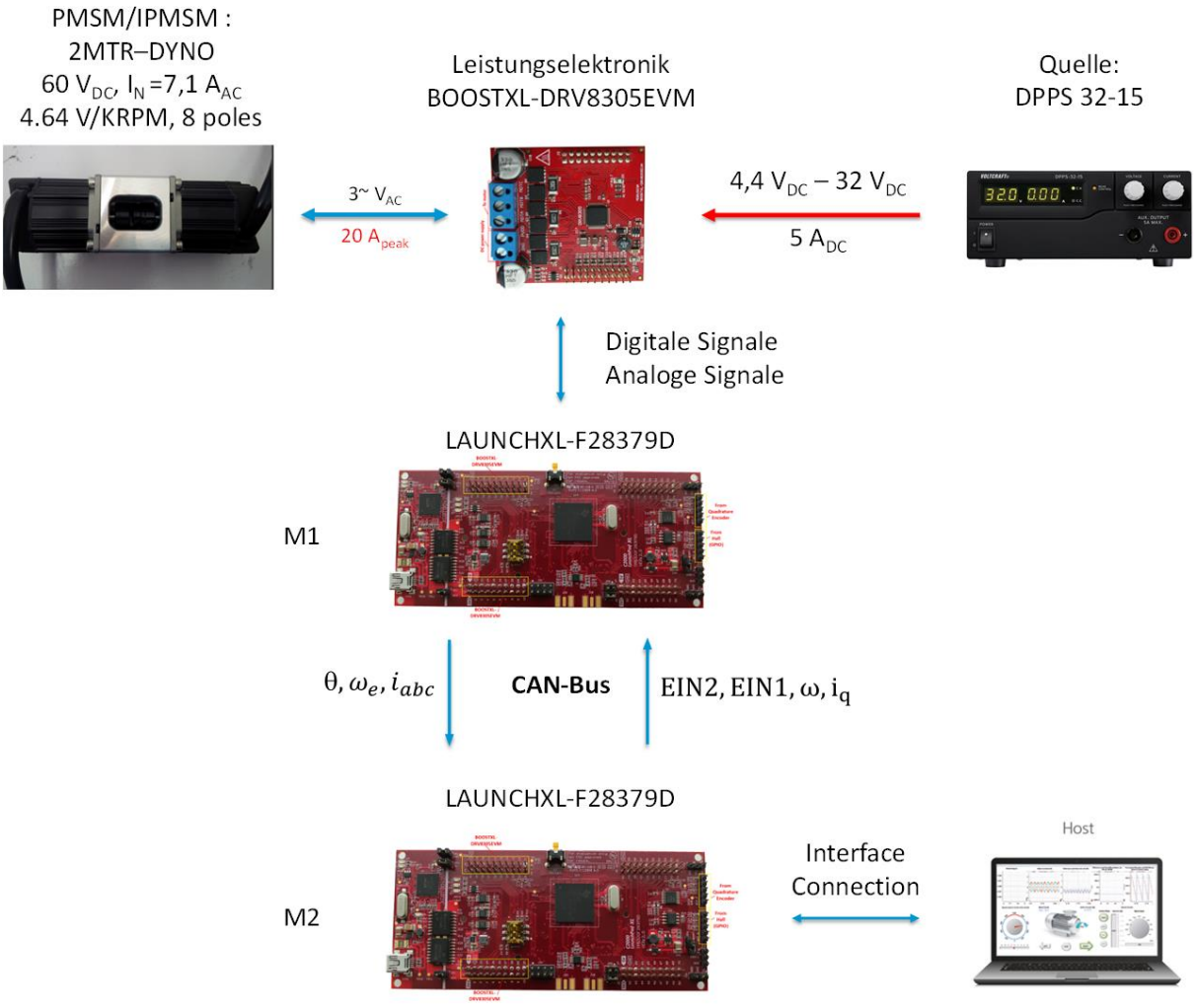

Abbildung 10: Blockschaltbild von System

In der obigen Abbildung 10 zeigt das Blockschaltbild an. In diesem Projekt wird der Elektromotor 2MTR-DYNO an einem Motorantrieb BOOSTXL-DRV8305EVM angeschlossen, eine 3-phasige, bürstenlose 15A-DC-Antriebsphase, die auf dem DRV8305 und CSD18540Q5B basiert. Der Antrieb wird von einer Quelle DPPS 60-15 eingespeist und wird mit einem TI Launchpad LAUNCHXL-F28379D-Entwicklungskits (M1) für ein komplettes Motorsteuerungssystem angedockt. M1 wird mit einem anderen Launchpad M2 über CAN-Bus gesteuert und gleichzeitig wird M2 mit einem Computer verbunden. Über den CAN-Bus kann der Benutzer den Motor ein- oder ausschalten, und die Motordrehzahl oder den Strom steuern.

Umgekehrt werden die Motorparameter wie Lagewinkel, Rotordrehzahl oder Ströme auf dem Computerbildschirm angezeigt.

3.2. Übertragende Daten

In diesem Projekt wird CAN-Bus 2.0B verwendet. Dabei beträgt die maximale Länge für Daten 8 Byte, was 64 Bit entspricht. Nachfolgend werden die Längen der übertragenden Parameter definiert.

Signal	Motorregelung M1	Steuerung M2	Beschreibung	Länge
EIN2	Empfänger	Sender	Einmal zu Beginn durführen von 0 auf 1 für ca. 10s dann auf 0	1 Bit
EIN1	Empfänger	Sender	Schaltet Motor ein Erst nach Offsetabgleich möglich 1 ein 0 aus	1 Bit
ω	Empfänger	Sender	Stellt die Motordrehzahl ein	32 Bit
i_q	Empfänger	Sender	Stellt den Strom Iq ein Default: 0	15 Bit
θ	Sender	Empfänger	Lagewinkel	16 Bit
ωe	Sender	Empfänger	Rotordrehzahl	32 Bit
i_{abc}	Sender	Empfänger	Motorstrom --> Drei Ströme	2 · 32 Bit = 64 Bit

Tabelle 1: Übertragenden Daten via CAN-Bus

4. Umsetzung

Nachdem die Daten definiert wurden und man weißt was sind die Nachrichten, die von M1 nach M2 und umgekehrt übertragen werden müssen, um die Kommunikation zwischen den zwei Microkontroller herzustellen, kann man die Buskommunikation mit verschiedenen Bausteinen in PLECS starten.

4.1. Programmieren

4.1.1. CAN Port

Der Baustein richtet einen CAN-Kommunikationsport ein. Der Eingang *en* bestimmt den Zustand des CAN-Ports. Das Setzen von *en* auf Null zwingt den CAN-Port in den Zustand Bus-Off, während das Setzen des Ports auf 1 dem CAN-Port ermöglicht, in den Zustand Bus-on überzugehen.

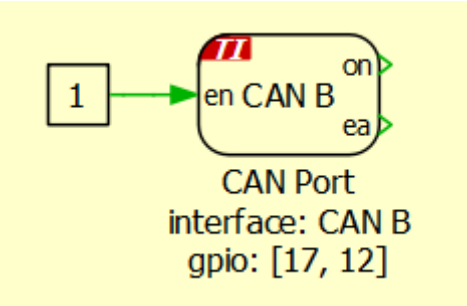

Abbildung 11: CAN Port

4.1.2. CAN-Transmit

Der Baustein CAN-Transmit sendet Daten auf einem CAN-Bus. Die zu sendenden Daten müssen am Bausteineingang *d* als vektorisiertes Signal mit dem Datentyp uint8 bereitgestellt werden. Die Länge der gesendeten CAN-Nachricht wird durch die Breite des Eingangssignals (1 bis 8 Byte) bestimmt.

Nachrichten werden entweder regelmäßig mit einer festen Abtastzeit oder auf Abruf gesendet, wenn sich der Triggereingang ändert. Bei Konfiguration für die getriggerte Ausführung werden Nachrichten gesendet, wenn sich das Trigger-Signal in der durch den Trigger-Typ-Parameter festgelegten Weise ändert.

4.1.3. CAN Pack

Der Baustein CAN-Pack nimmt PLECS-Signale als Eingaben und generiert eine CAN-Nachricht.

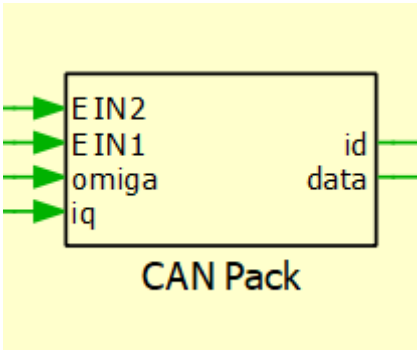

Abbildung 12: CAN Pack

Ein Signal in einer CAN-Nachricht wird durch seinen Datentyp, seine Byte-Reihenfolge (Little Endian / Big Endian) und sein Startbit und seine Länge innerhalb der CAN-Nachricht definiert. Eine Nachricht kann mehreren Signalen beinhalten so lange die Gesamte Länge 8 Byte nicht überschreitet. Ein Signal kann skaliert und ein Offset angewendet werden, um Fließkommasignale effizient als ganze Zahlen zu senden. Die Bits innerhalb der CAN-Nachricht sind von 0 (erstes Bit des ersten Bytes) bis 63 (höchstwertiges Bit des letzten Bytes) nummeriert.

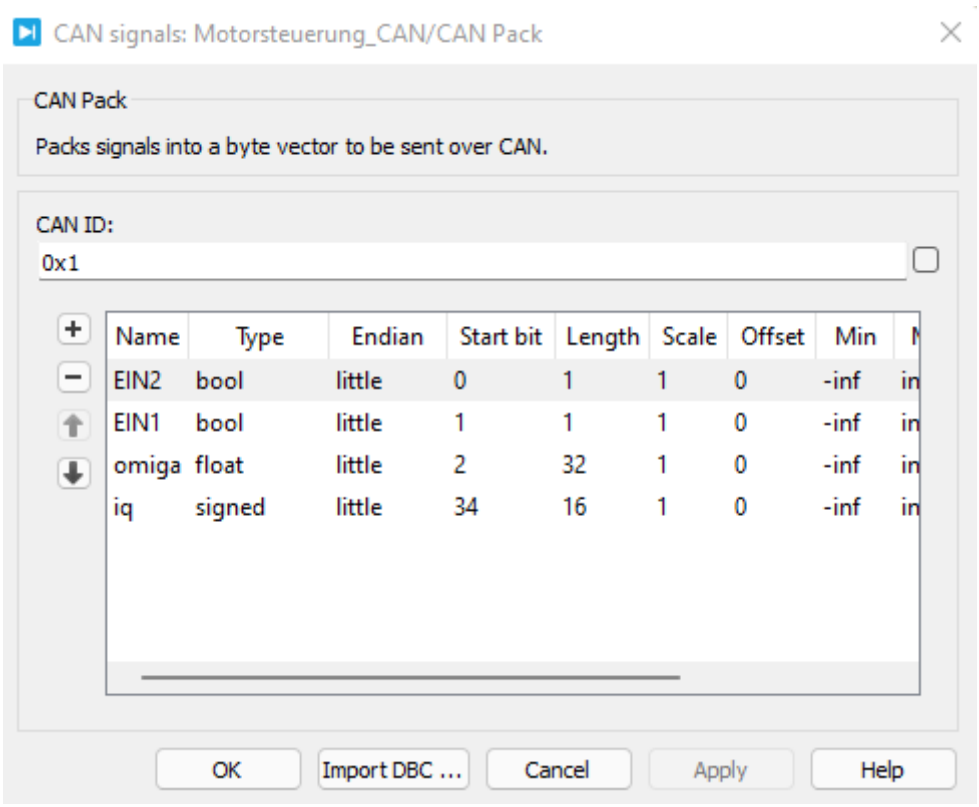

Abbildung 13: Nachrichtdefinition in PLECS

Daten, die für die Steuerung der Reglung gebraucht werden. werden in der Tabelle in Abbildung 13 als Beispiel dargestellt.

Der Baustein CAN-Transmit sendet Daten auf einem CAN-Bus. Die zu sendenden Daten müssen am Bausteineingang *d* als vektorisiertes Signal mit dem Datentyp uint8 bereitgestellt werden. Die Länge der gesendeten CAN-Nachricht wird durch die Breite des Eingangssignals (1 bis 8 Byte) bestimmt.

Nachrichten werden entweder regelmäßig mit einer festen Abtastzeit oder auf Abruf gesendet, wenn sich der Triggereingang ändert.

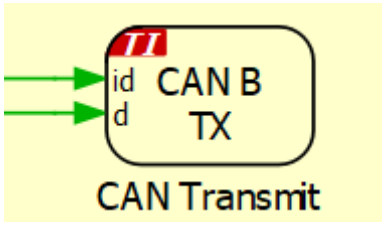

Abbildung 14: Nachrichtdefinition in PLECS

Der Baustein CAN-Receiver initiiert den Empfang von CAN-Nachrichten mit dem angegebenen Identifier (ID) auf der angegebenen CAN-Schnittstelle. Beim Empfang einer CAN-Nachricht werden die Daten am Blockausgang *d* als vektorisiertes Signal, der bereitgestellten Rahmenlänge zur Verfügung gestellt wird. Der Ausgang *v* ist für einen Simulationsschritt 1, wenn neue Daten empfangen werden, ansonsten 0.

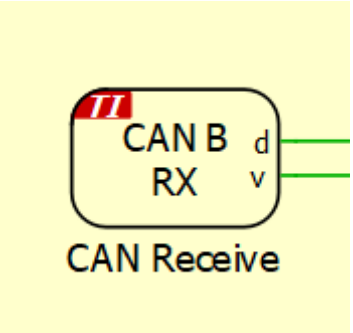

Abbildung 15: CAN-Receiver

Der Baustein CAN-Unpack generiert PLECS-Signale aus einer CAN-Nachricht.

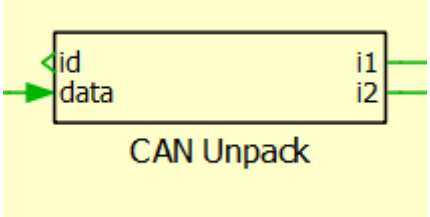

Abbildung 16: CAN-Unpack

Anhand der vorgestellten Bausteine kann die Kommunikation hergestellt werden. Zuerst werden die Informationen für die Steuerung von M1 gesendet

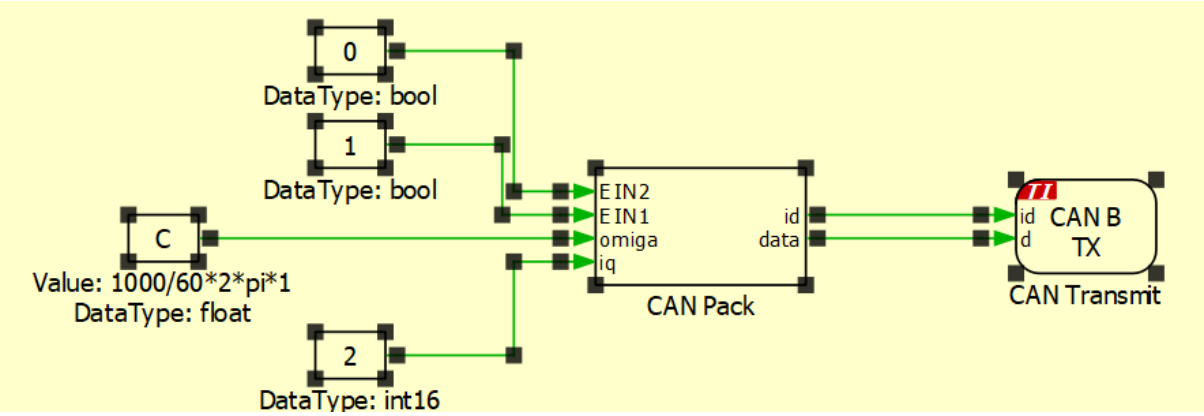

Abbildung 17: Daten für die Steuerung (Sendvorgang)

Und danach in M2 empfängt.

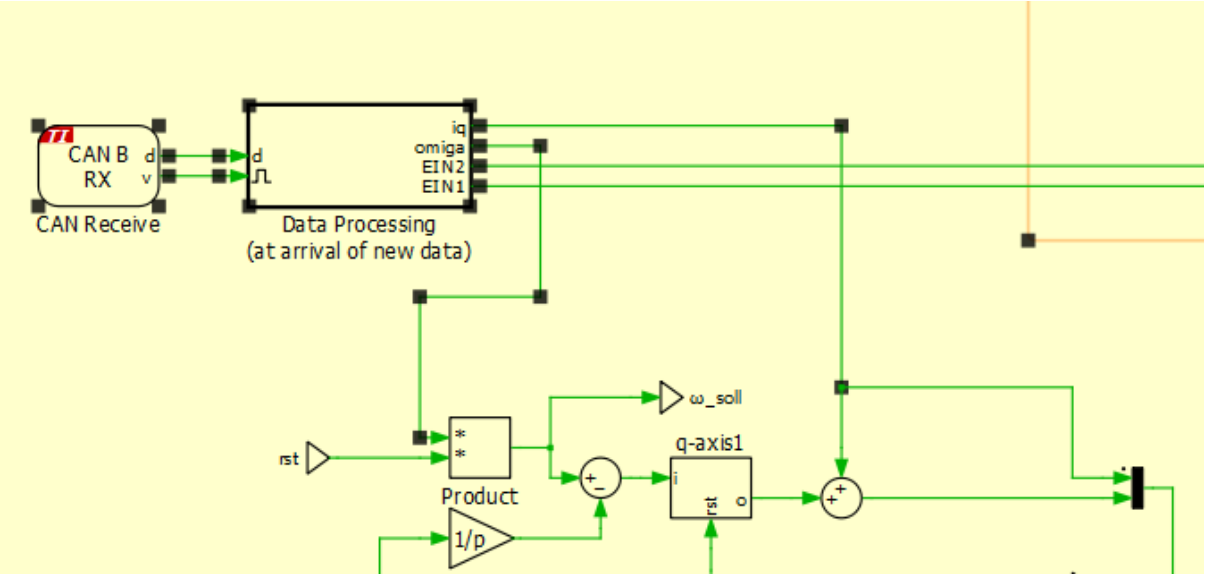

Abbildung 18: Daten für die Steuerung (Empfangsvorgang)

Die Microkontroller M1 muss auch Daten empfangen, die für weitere Verarbeitung oder Analysen gebraucht werden. Da CAN-Nachricht maximal 8 Byte sein darf, mussten die Daten an zwei Nachrichten geteilt werden. Winkel, Drehwinkle und Phasenströme werden von M2 zu M1 gesendet.

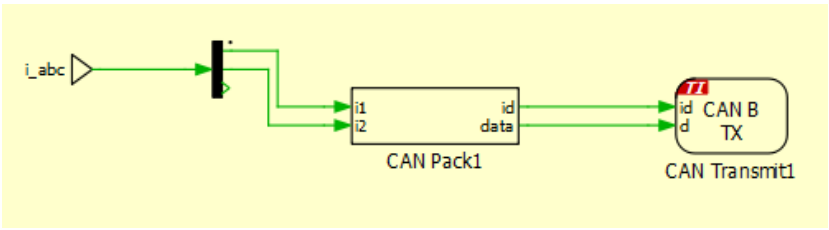

Abbildung 19: Daten für die Verarbeitung (Sendvorgang)

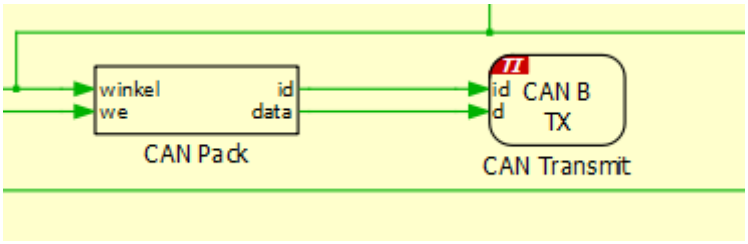

Abbildung 20: Daten für die Verarbeitung (Sendvorgang)

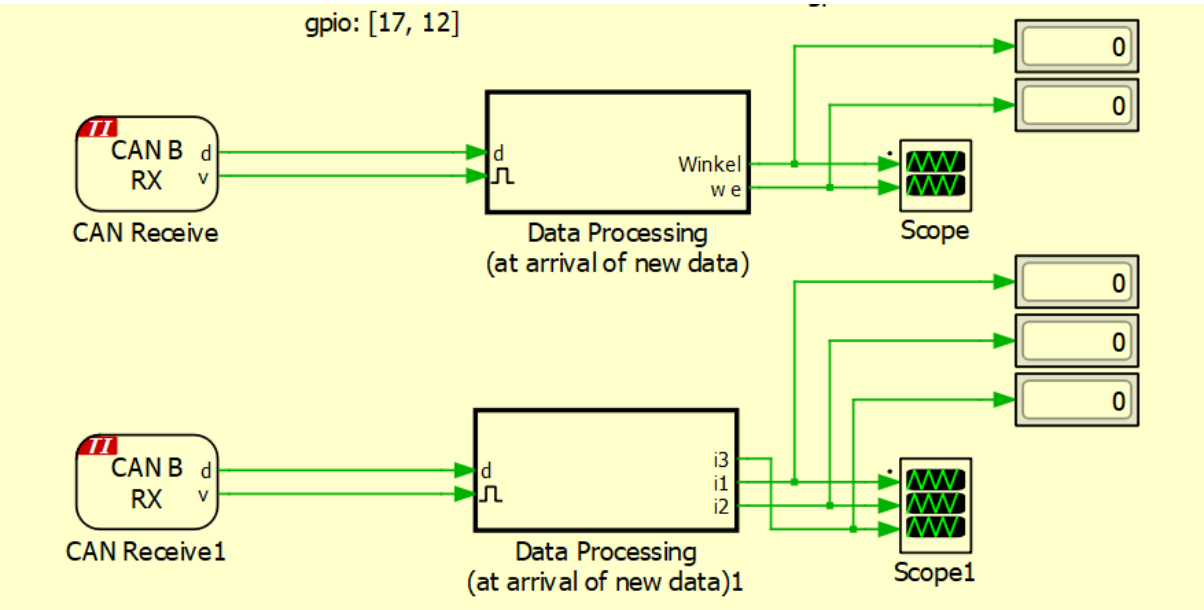

Abbildung 21: Daten für die Verarbeitung (Empfangsvorgang)

4.2. Der Programms

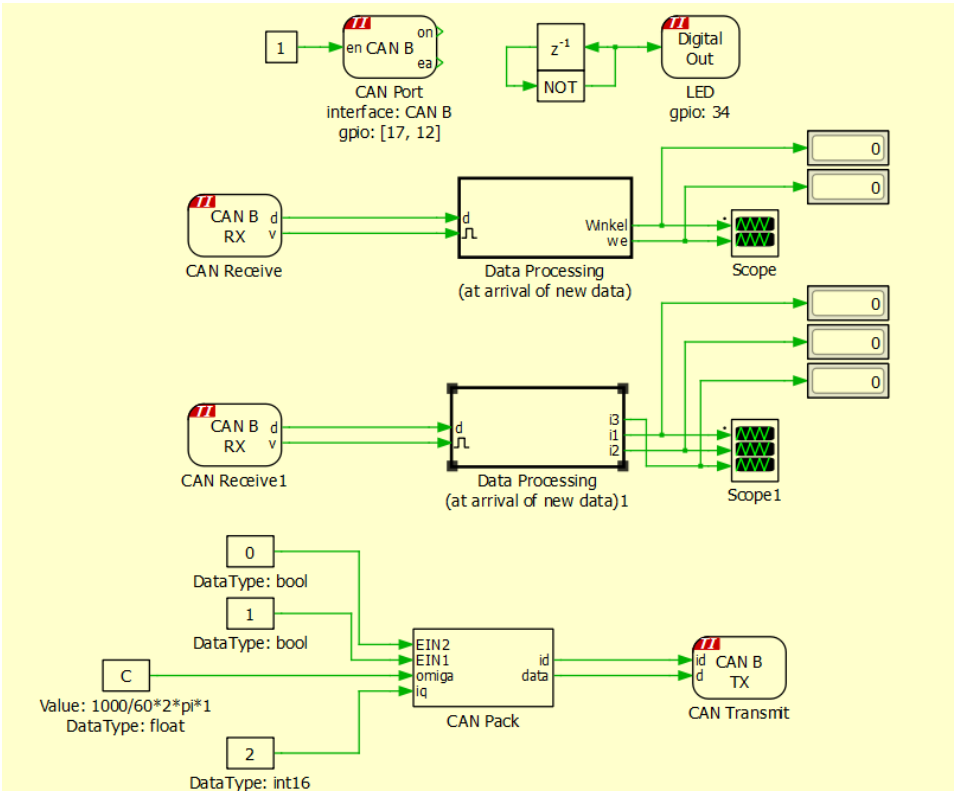

Abbildung 22: Gesamtprogramm für den Steuerung-Microkontroller

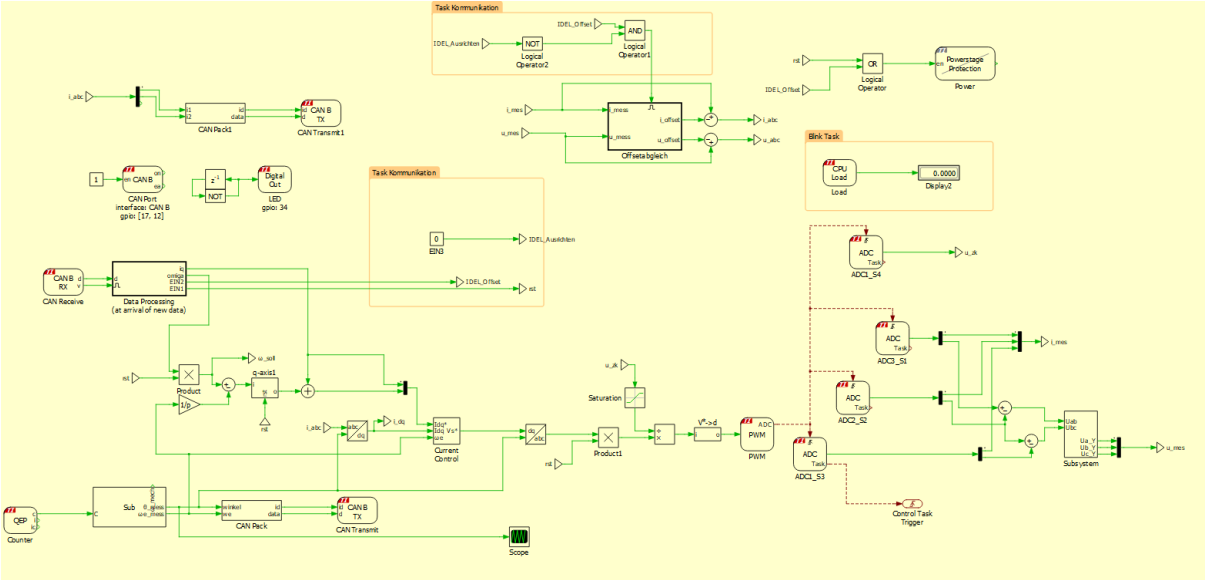

Abbildung 23: Gesamtprogramm für den Reglung-Microkontroller

4.3. Ergebnisse des Tests

Nachdem die Programme jeweils auf die Microkontroller installiert werden, kann das Gesamtsystem mit bestimmtem Ablauf getestet werden. Zunächst wird der *EIN2* auf „1“ gesetzt, um der Versatzwinkel im Motor auszugleichen. Nach ca. 10s ist der Versatzwinkel im Motor ausgeglichen, kann der *EIN2* auf „0“ zurückgesetzt. Der *EIN1* kann erst nach dem Offsetabgleich den Motor einschalten, wenn er auf 1 gesetzt ist und auf 0 umgekehrt. Wenn man den Motor eingeschaltet hat, kann der Drehwinkel des Motors mit *omiga* eingestellt werden. Mit *iq* kann man den Strom des Motors einstellen. Die Bilder unter sind die Ergebnisse in den 2 Scope des Tests.

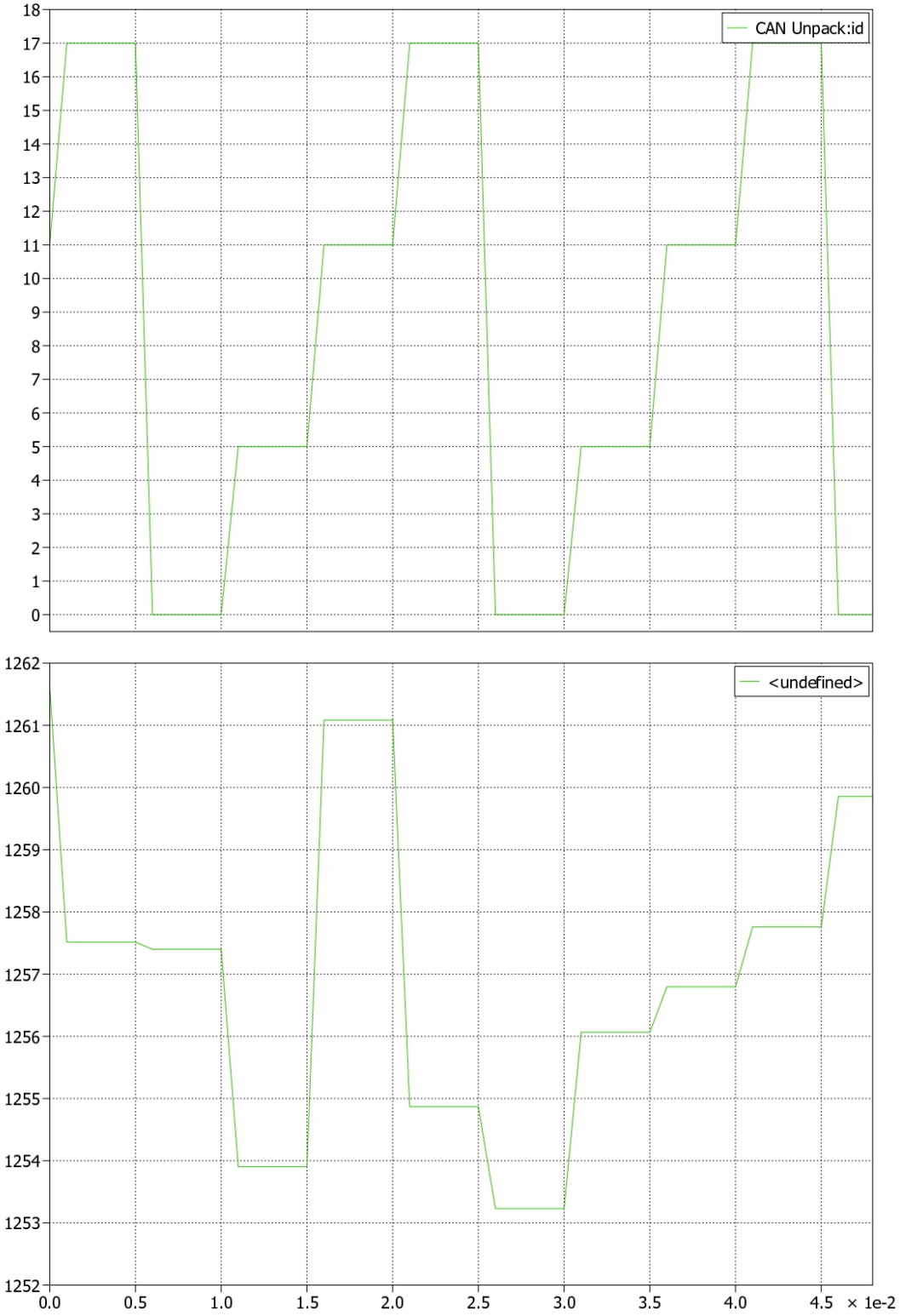

Abbildung 24: Lagewinkel und Rotordrehzahl mit der eingestellten Drehzahl von 500 U/min

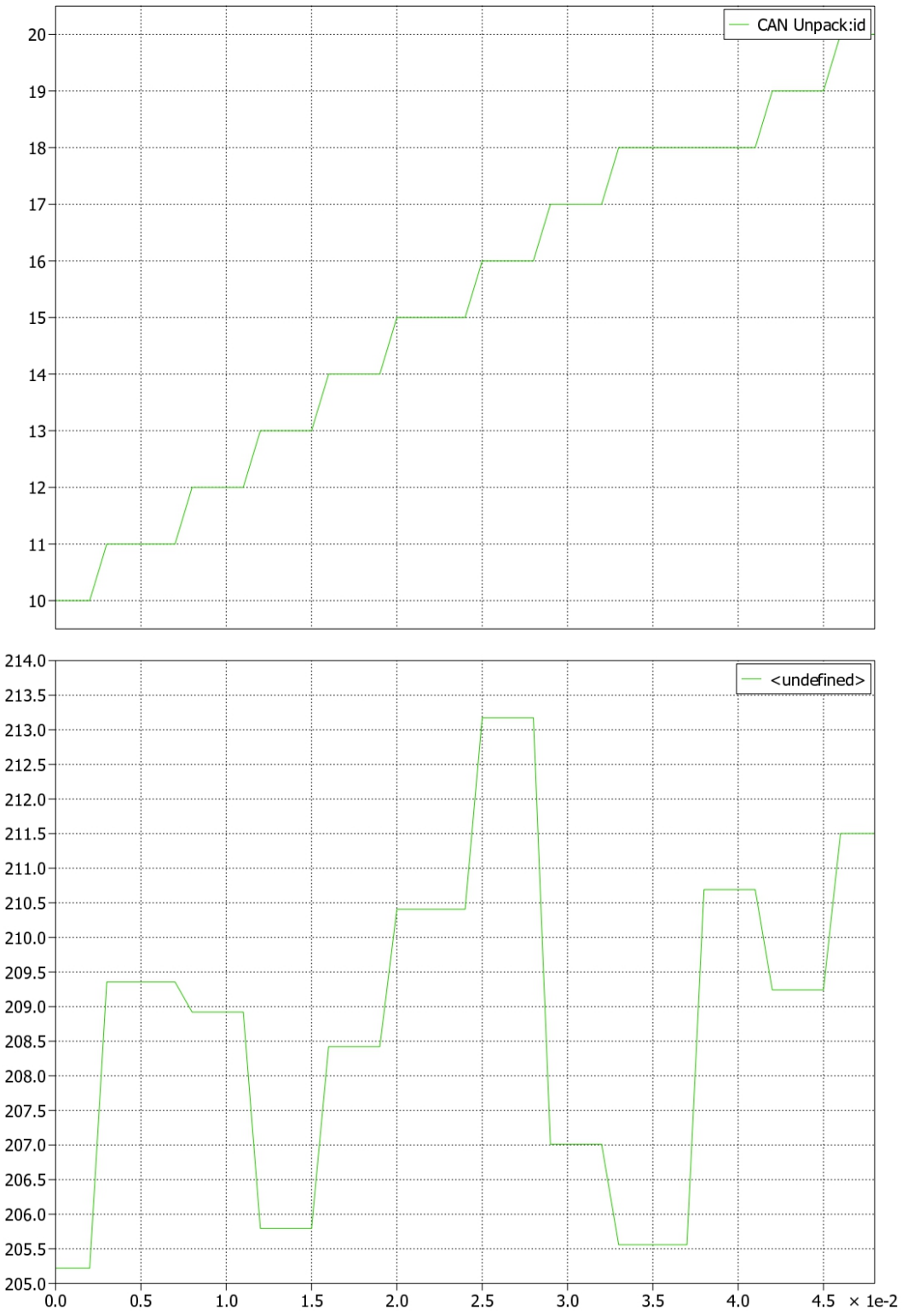

Abbildung 25: Lagewinkel und Rotordrehzahl mit der eingestellten Drehzahl von 1000 U/min

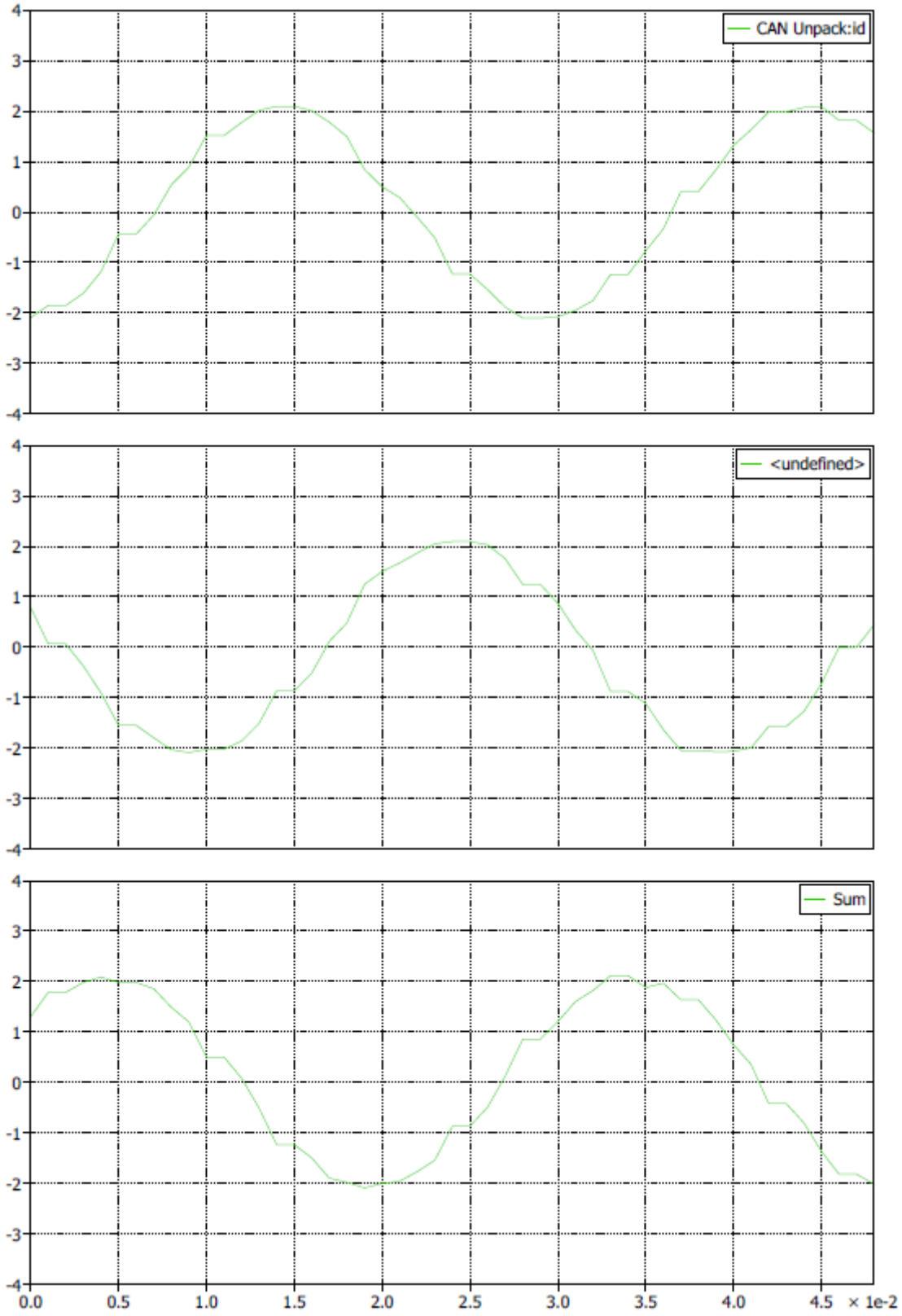

Abbildung 26: Phasenströme bei der eingestellten Drehzahl von 500 U/min

Wählt man eine möglichst größere Drehgeschwindigkeit, wird die Fähigkeit des Buses übergestiegen, der sinusförmige Verlauf kommt verzehrt an (Abbildung 27).

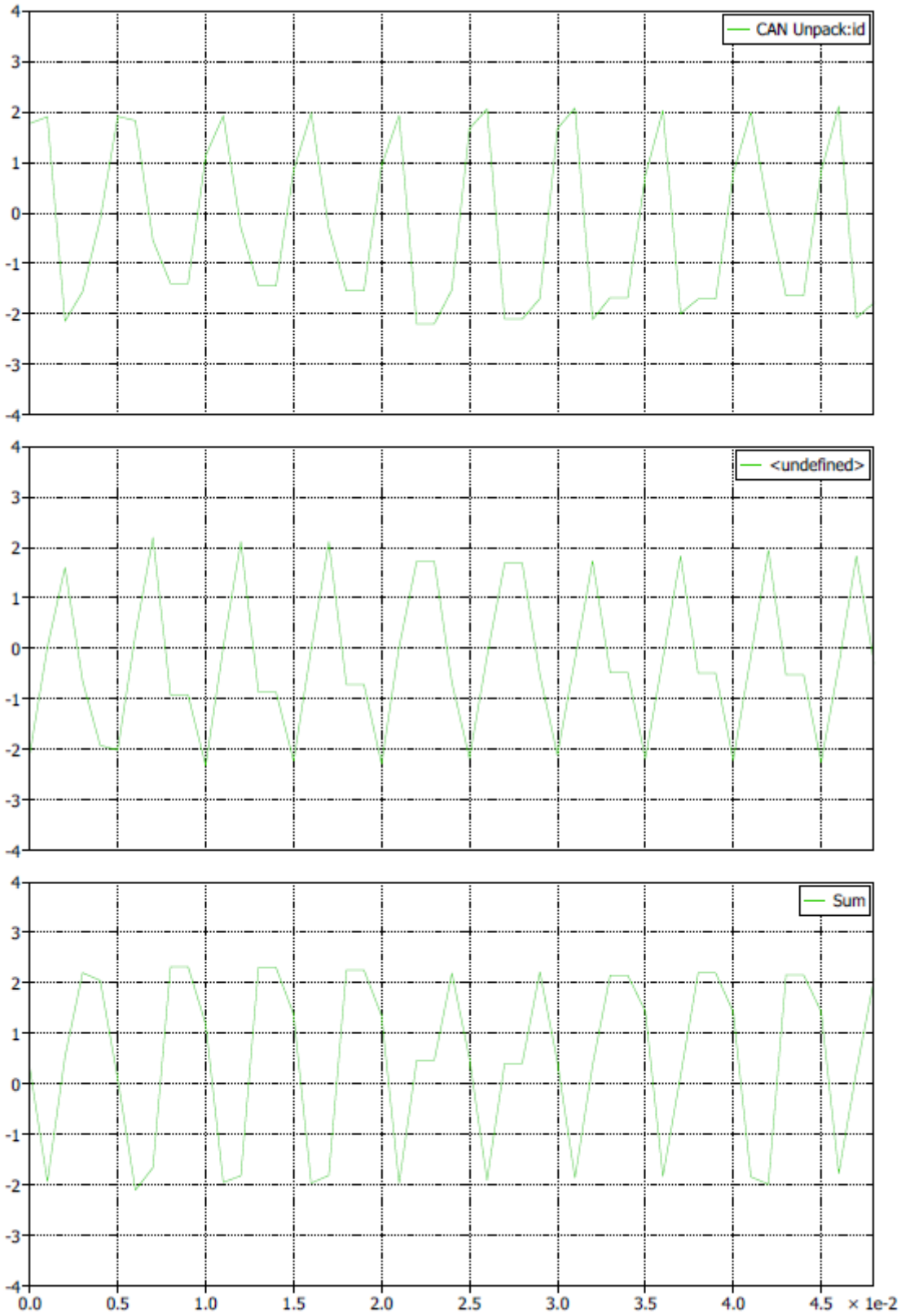

Abbildung 27: Phasenströme bei der eingestellten Drehzahl von 1000 U/min

5. Fazit

In diesem Projekt sollte das Wissen über den CAN-Bus weiter vertieft und umgesetzt werden. Dabei haben wir die Möglichkeit, die Kenntnisse mit PLECS zu erlernen und zu üben. Dies hilft uns, eine andere Methode für die Implementierung der Hardware kennenzulernen.

Mit Hilfe des Betreuers und der Theorien aus der Vorlesung “Grundlagen des Projektmanagements” kann man die Projektprozess sowie die Risiken der Arbeit von Anfang definieren und kontrollieren. Der Zeitplan für die Bearbeitung der Arbeit spielt eine große Rolle, um das Projekt in richtiger Zeit abzuschließen. Die Teamarbeit wird auch verstärkt und verbessert.

6. Quelle

- PLECS: https://www.plexim.com/products/plecs
- Texas Instruments C2000 LaunchPad: https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/c2000-real-time-control-mcus/overview.html
- M. Sc. Michael Brüns, „Kick-off Mikrocontrolleranwendungen“, 2022
- Controller Area Network: https://de.wikipedia.org/wiki/Controller_Area_Network