CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

Begleitende Projekte
CAN Bus

- CAN Entwicklungsboard
  Konzeption, Umsetzung

- Thomas Böhm
- Matthias Hirschberger
- Kevin Oswald
- Michael Wipfler

Inhalt

- Einführung CAN Bus
- CAN Interfaces am Markt
- Aufgabenstellung
- Projektkonzept
- Ausführung
  - Komponentenübersicht
  - Schaltplan und Layout
  - Software

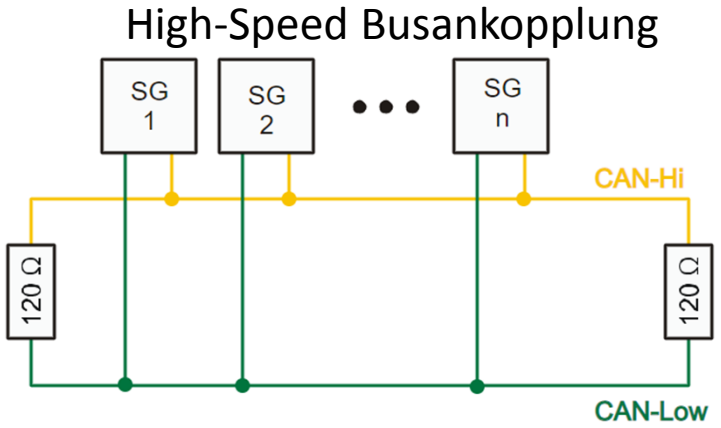
Grundlagen

Controller Area Network (CAN)

- Bitserieller Datenbus
- Datenrate bis 1 Mbaud, bis zu 8 Mbit/s bei CAN-FD
- Automotive und industrielle Anwendungen
- Hohe Zuverlässigkeit
- Entwickelt von Bosch ~ 1983 (Intel 82526)
- Aktuelles Protokoll: CAN 2.0 A, 2.0 B ~ 1995
- Time-Triggered CAN: TTCAN, ~ 2000
- CAN FD: Flexible Data-Rate, seit ~ 2011

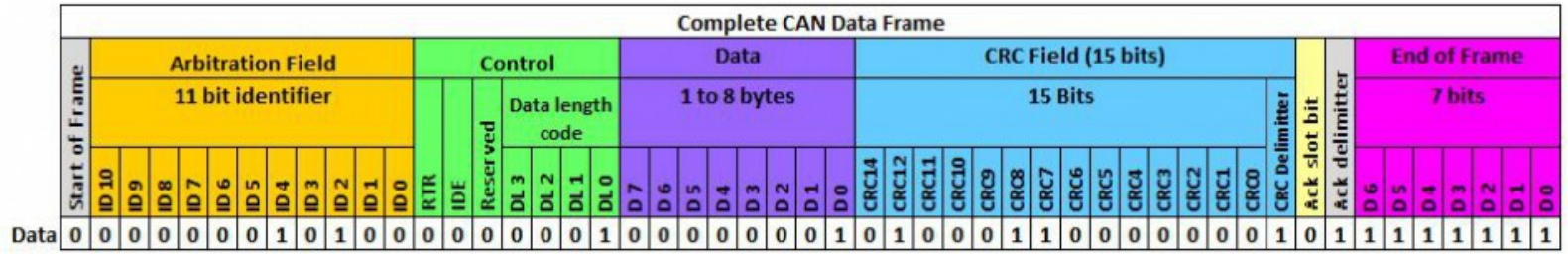
Nachrichtenübertragung & Topologie

- Kennzeichnung von Botschaften mittels Identifier
- Priorität durch Identifier festgelegt
- Rundfunk – Prinzip → jeder Empfänger bekommt jede Nachricht
- 2 verdrillte Datenleitungen – CAN High und CAN Low
- Beginn und Ende mittels 120Ω Abschlusswiderständen „abgeschlossen“



Frame Arten

- Daten-Frame → Transport der Daten
- Remote-Frame → Anforderung von Daten-Frames eines anderen Teilnehmers
- Error-Frame → signalisiert allen Teilnehmern erkannte Störung
- Overload-Frame → „Zwangspause“ zwischen Daten- und Remote-Frames



CAN Interfaces am Markt

- IXXAT
  - USB-to-CAN V2, USB-to-CAN FD, SimplyCAN
- Vector
  - VN1630A, VN7610, …
- Vorteile
  - Standardisiert
  - Support
  - Umfangreiche Softwarepakete
- Nachteile
  - Teuer in der Anschaffung (Lizenz)
  - Wartungskosten
  - Nur bedingt individualisierbar

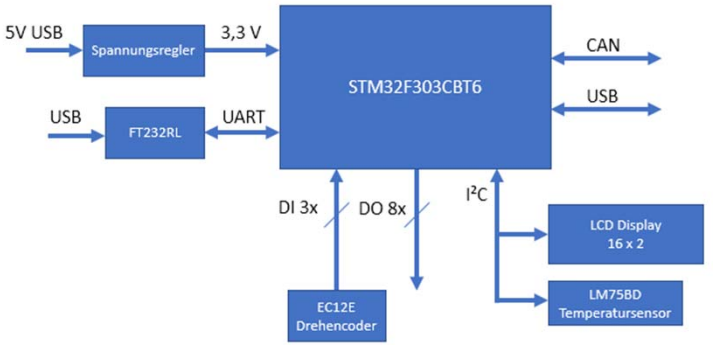
Projekt:
CAN Entwicklungsboard

Aufgabenstellung

- Komponentenauswahl
- Entwurf eines Schaltplanes
- Design eines PCB-Layouts
- Fertigung des Entwicklungsboards
- Erstellung einer PC-Benutzeroberfläche in C#
- Erstellung eines µController Programms
- Definition der Kommunikation zwischen µController und C# Programm

Projektkonzept

- 2 Boards
- Board 1:
  - Temperaturmessung (über I2C angebunden)
  - Anbindung an PC via USB (Virtuelle Serielle Schnittstelle)
  - 8x Digital out auf LEDs
  - 3x Digital in über Drehencoder
  - Übertragung von Temperatur und Schaltbefehle von PC via CAN an Board 2
- Board 2:
  - LCD (über I²C angebunden)
  - 8x Digital out auf LEDs
  - 3x Digital in über Drehencoder
  - Empfang von Temperatur und Daten + Darstellung auf Display

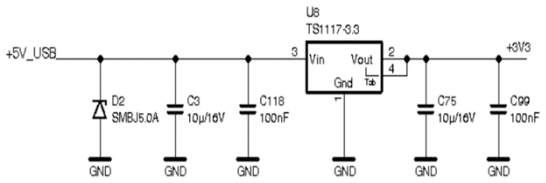


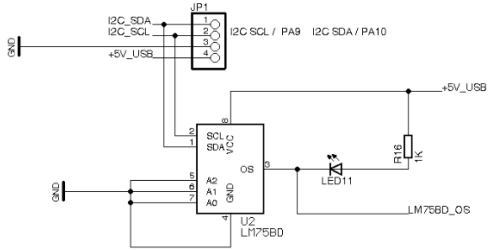
Komponenten

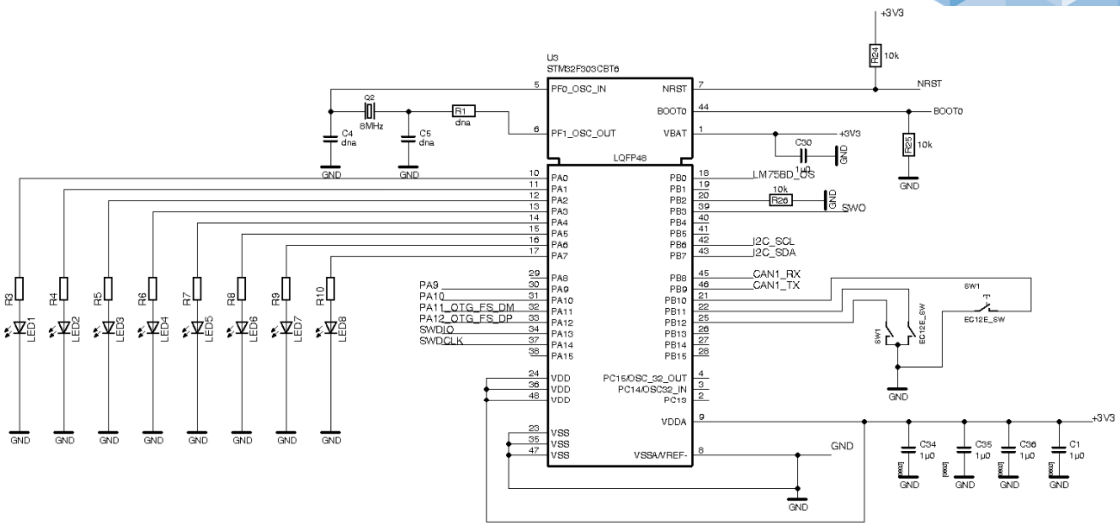
- µ-Controller: STM32F303CBT6 Cortex M3
- LCD Display 16x2 Zeichen (I^2C)
- LM75BD Temperatursensor (I^2C)
- EC12E Drehencoder
- FT232RL (USB to Serial Converter)
- TS1117-3.3 (Längsregler)
- Diverse Teile wie LEDs, Widerstände, Kondensatoren, Gleichtaktdrossel, Dioden und Stecker

Schaltplan Teil 1

- Spannungsversorgung
- STM32F303CBT6
- Anzeige- und Eingabeelemente

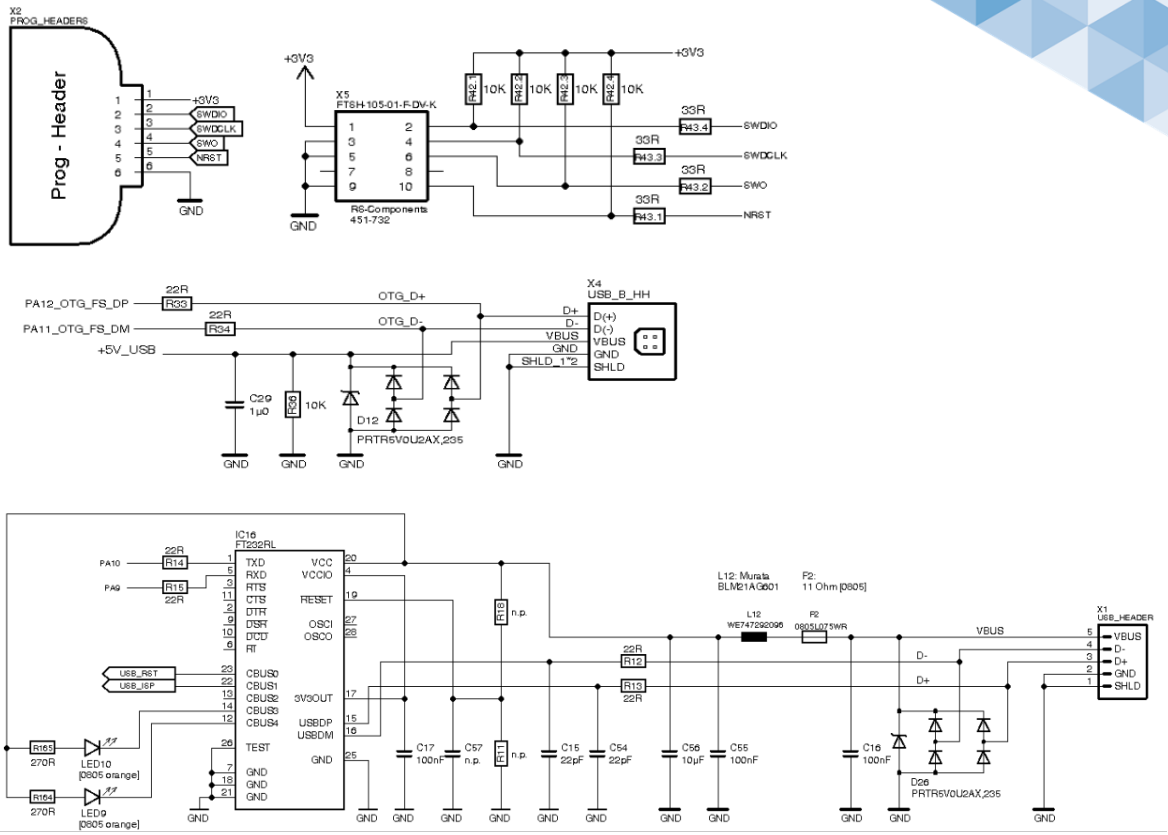






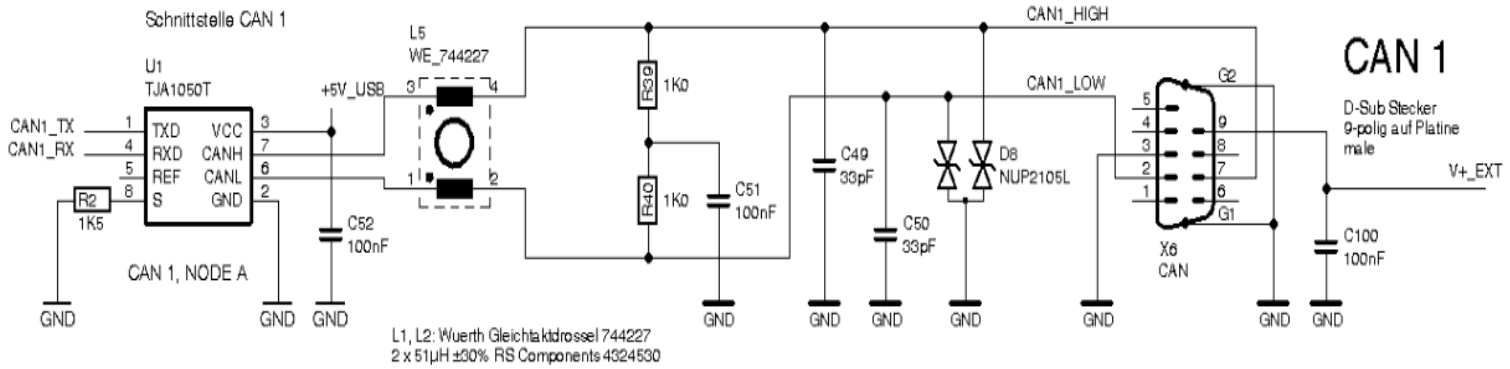
Schaltplan Teil 2

- Debug-Schnittstellen
- USB-Schnittstelle des Mikroprozessor
- USB-UART (FTDI) Umsetzer



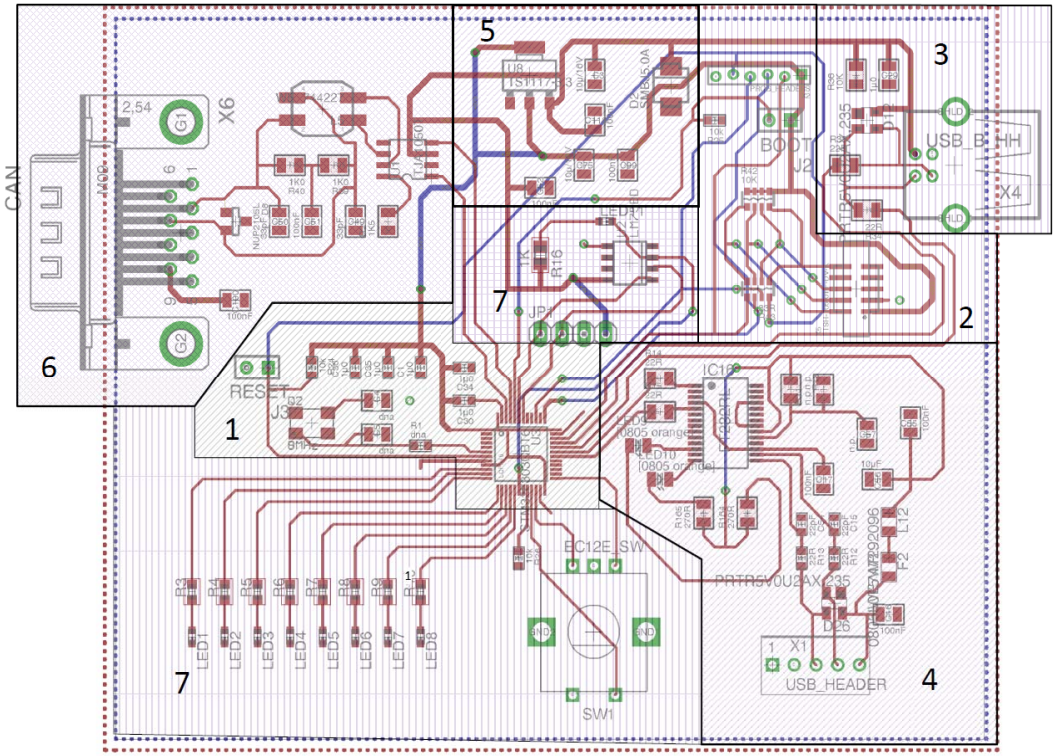
Schaltplan Teil 3

- CAN-Bus Schnittstelle
  - Pinbelegung: CAN in Automation (CiA)



Layout

- STM32 µ-Controller (1)
- Debug-Schnittstellen (2)
- USB-Schnittstelle des Mikroprozessor (3)
- USB-UART (FTDI) Umsetzer (4)
- Stromversorgung (5)
- CAN-Bus Schnittstelle (6)
- Anzeige- und Eingabeelemente (7)

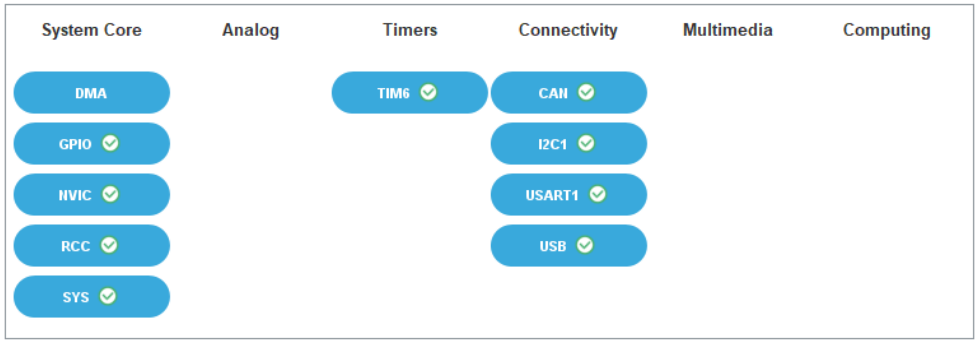


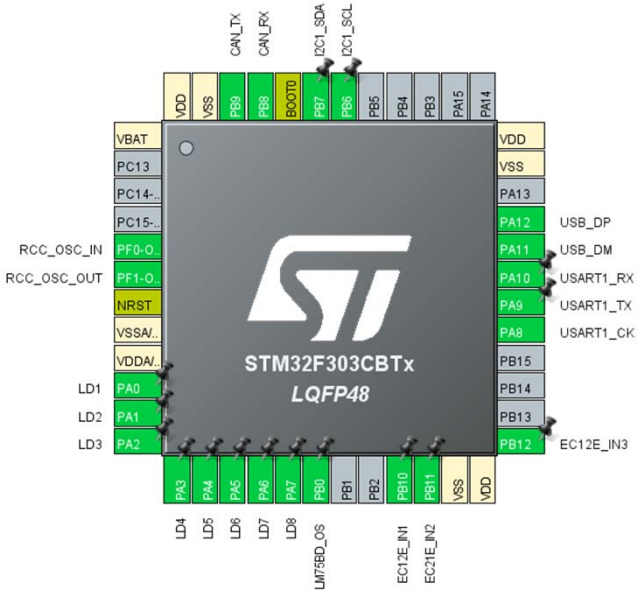
Software – µ -Controller

- Entwicklungsumgebung: STM32CubeIDE
- Hardwarekonfiguration: CubeMX
- Erstimplementierung auf NUCLEO-F413ZH Development Board
- Entwicklungsbegleitende Tests mittels
  - Vector VN1630A CAN Interface
  - Vector CANalyzer V13
  - HTerm
- Danach Portierung auf STM32F303CBTx

Pinout & Konfiguration

Aktivierte Komponenten:





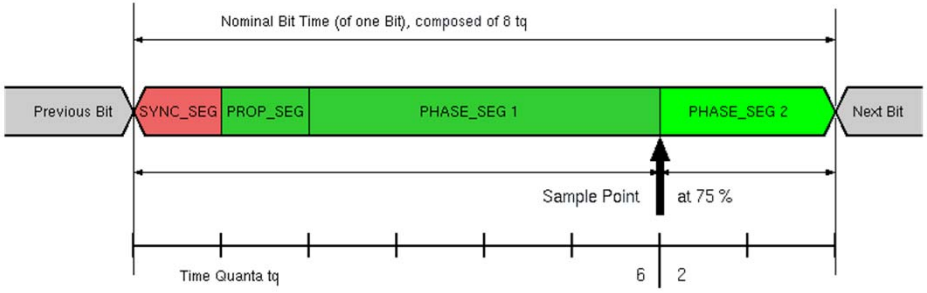
- Generierung des Codegerüsts nach erfolgter Pin- & Komponentenkonfiguration

Softwarearchitektur

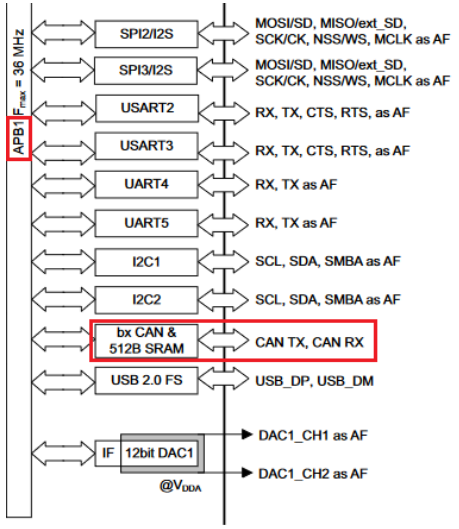
- **Main**
  - Initialisierung der Komponenten
  - Start d. Interrupts (UART, TIM, CAN)
  - Infinite Loop
    - Check for new valid UART Msg
- **HAL_UART_RxCpltCallback**
  - Aufruf sobald 1byte empfangen wurde
  - Empfangs-Statemachine
- **UART_CAN_Msg_RxCplt_Handler**
  - Nachricht am CAN senden
  - Digital Out schalten
- **HAL_TIM_PeriodElapsedCallback**
  - Aufruf alle 1sec
  - I²C Temp. Lesen
  - Temp. Senden (UART & CAN)
  - Temp auf LCD aktualisieren
- **HAL_CAN_RxFifo0MsgPendingCallback**
  - Aufruf sobald neue Nachricht in CAN FIFO0
  - Digital Out schalten
  - Nachricht auf UART senden

CAN-Timing

- Osc-Clock (8MHz) * PLLMul (6) * ABP1 Prescaler (/2)
- APB1 Peripheral Clock: 24MHz
- Time Quantum Prescaler: 15
- Time Quanta: 16 (BitSeg 1 = 13TQ, BitSeg2 = 2TQ)
- CAN-Bitrate = 24MHz / 15 / 16 = **<u>100kbit/s</u>**



Quelle: bittiming.can-wiki.info



Quelle: st.com – Datenblatt STM32F303BC

```
hcan.Init.Prescaler = 15;
hcan.Init.TimeSeg1 = CAN_BS1_13TQ;
hcan.Init.TimeSeg2 = CAN_BS2_2TQ;
```

Protokoll µC <-> PC

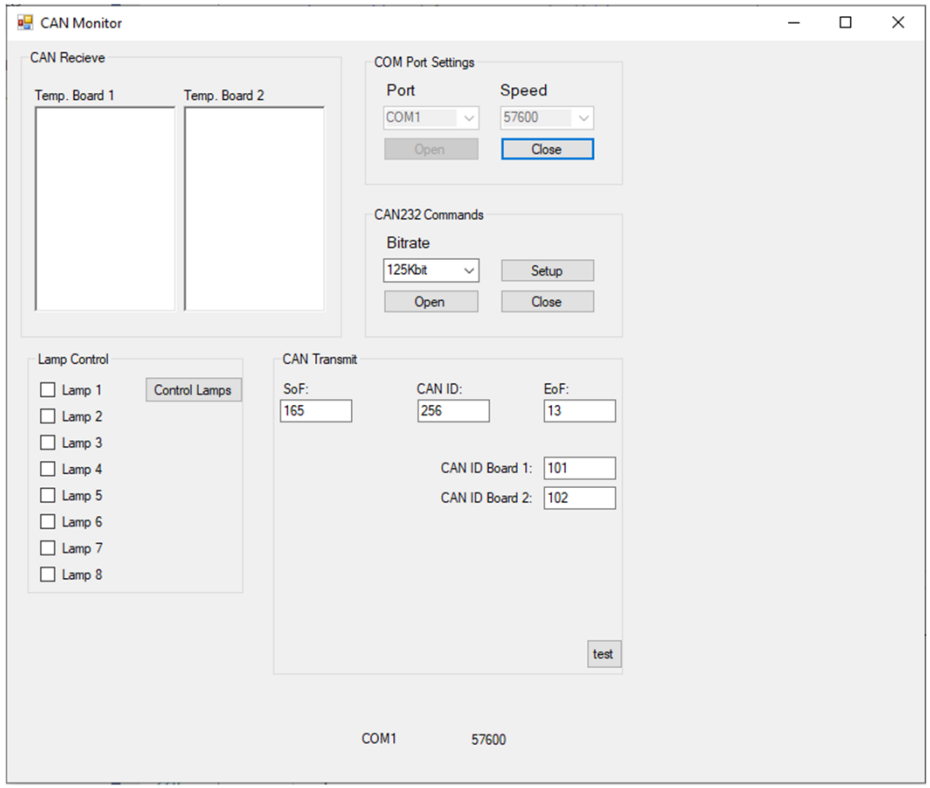
- CAN-Protokoll nachempfunden
- 11bit CAN-ID als Identifier
- Max 8 Datenbytes
- Temperaturauflösung: 0.1 °C

Conversion:
°C = (RAW / 10) - 40
RAW = (°C + 40) * 10

Message Name	Temperature Board 1									
Sender	Board 1									
Receiver	PC									
Frame	**SOF**	**ID**		**LEN**	**DAT**					**EOF**
Description	Starf of Frame	CAN-ID		Length [Bytes]	Temperature [(°C / 10) - 40]					End of Frame
Len (Bytes)	1	2		1	1-8					1
Data (DEZ)	165	101		2	650					13
Data (HEX) - Combined	A5	65		2	28A					D
Data (HEX) - Splitted	0xA5	0x00	0x65	0x02					0x02	0x8A	0x0D
HEX-String	A5 00 65 02 02 8A 0D									

Software – PC-Anbindung

- COM-Port Auswahl
- Temperaturauslese
- Lampensteuerung
- Testumgebung



Software – PC-Anbindung

- Flexibilität bei Einstellung der SoF, EoF und CAN-IDs
- Temperatur über die CAN-IDs der Boards in der Ausgabe entsprechend zuweisen
- Temperaturbereich: -40°C bis +100°C
- 8 Lampen getrennt voneinander steuerbar. Codierung Binär und Übergabe an den Mikrocontroller mittels einem Hexadezimal-Wert


Gibt es noch Fragen?
CAMPUS
02
Vielen Dank für die Aufmerksamkeit!