CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

BEGLEITENDE PROJEKTE CAN BUS

Vergleich CAN-Standard 2.0x mit CAN FD und Analyse der Software eines CAN-BUS Knotens

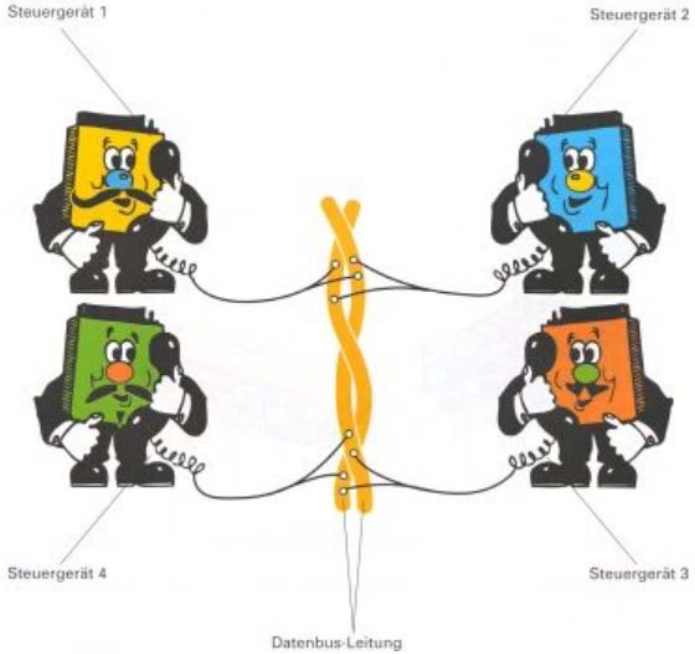
Daniel Pessler
Christoph Schwab

Inhalt

- Controller Area Network
- Unterschiede zwischen CAN 2.0 und CAN FD
- Verbesserungen und Vorteile
- Verbesserungen und Vorteile anhand eines Beispiels
- Praxisbeispiel: Analyse der Software eines CAN Bus Knotens

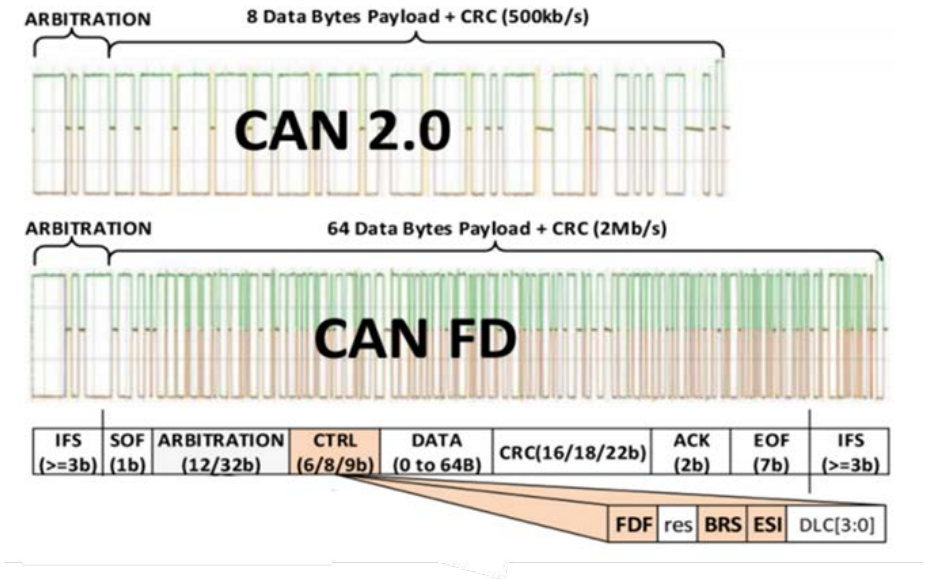
Controller Area Network

- 1983 von Bosch entwickelt
- Anwendungen:
  - →PKW
  - →Flugzeuge
  - →Roboter
  - →intelligente Steuerungen
  - →…



Unterschiede zwischen CAN 2.0 und CAN FD

- Anforderungen
- Nachrichtenrahmen
- Physical Layer
- CRC-Prüfung



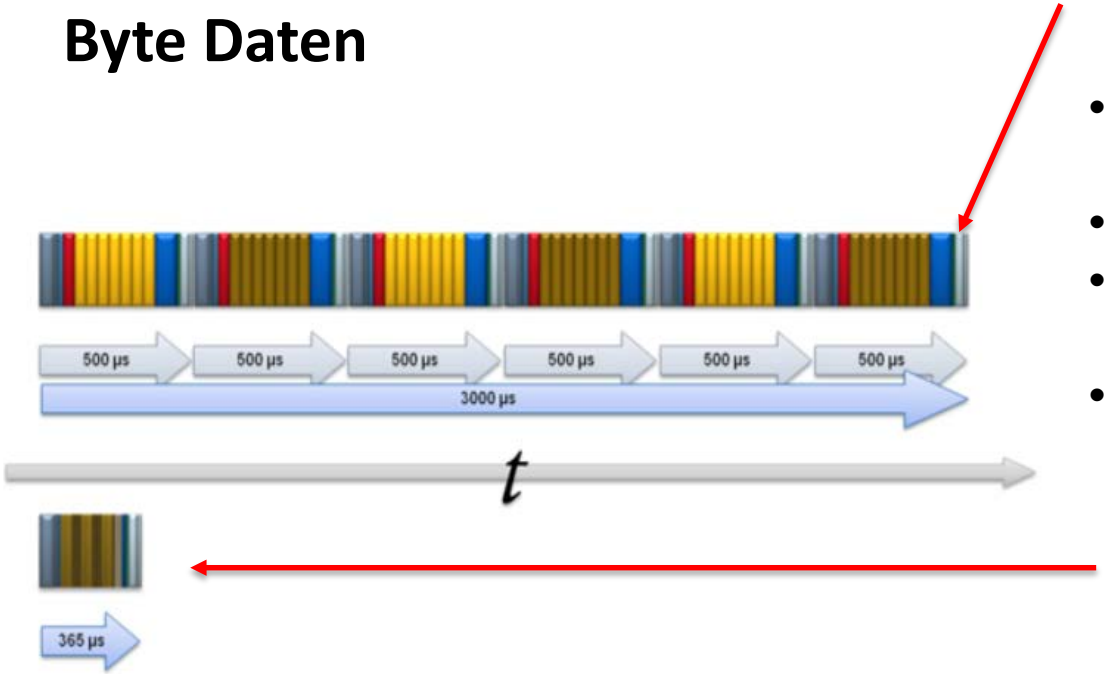
Verbesserungen und Vorteile

- Erweiterung der Nutzdaten
  → Effizientere Transportprotokolle

- Steigerung der Datenrate
  → Im Datenteil bis zu 15 Mbit/s möglich

- Verbesserte Fehlererkennung
  → Zusätzliches ESI Bit
  → Stuff-Bit Counter

Verbesserungen und Vorteile – Beispiel: Übertragung von 42/48 Byte Daten



Daten über klassischen CAN

- Arbitrierungs- und Datenrate 250kBit/s
- 8 Byte Nutzdaten (davon 1 Byte für Verwaltung)
- Max. mögl. Anzahl an Stuffbits
- 500 µs pro Nachricht, 3 ms für die gesamten Daten
- 42 Byte effektive Nutzdaten

Daten CAN FD

- Arbitrierungsrate 250 kBit/s
- Datenrate 2 Mbit/s
- Max. mögl. Anzahl an Stuffbits
- 365 µs für die gesamten Daten
- 48 Byte effektive Nutzdaten

Praxisbeispiel
Analyse der Software eines CAN BUS Knotens

- Aufgabenstellung
- Beschreibung der Hardware
- Konfiguration der Hardware

  → „Wie setze ich einen CAN Bus Controller in Betrieb“

Analyse der Software eines CAN BUS Knotens
Aufgabenstellung

- Betrachtung der Konfiguration (Hardware/Software)
  - Beschreibung des Evaluierungsboards
  - Studium der Datenblätter
- Einstellungen bei Inbetriebnahme eines CAN BUS
- Betrachtung eines SLIO äquivalenten Befehlssatzes

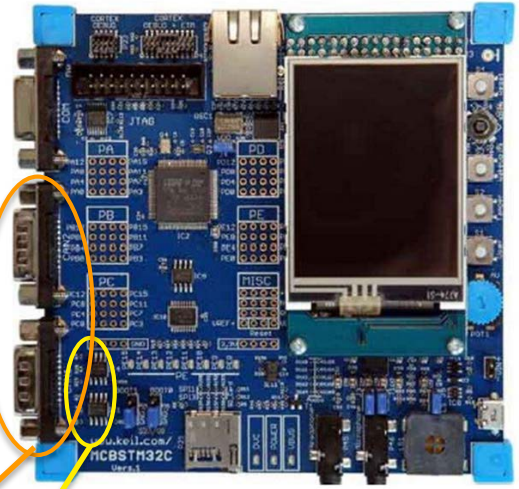
Ziel: Analyse und Beschreibung der Funktion eines vorliegenden CAN BUS Knotens

Beschreibung der Hardware Evaluierungsboard MCBMSTM32C

µController-Board der Firma Keil

Technische Spezifikation:

Größe:	115x115mm
Display:	LED, 240x320 pixel screen
Prozessor:	ARM Cortex M3
RAM:	64kB
FLASH:	256KB
I/O Port LEDs:	8
Power Supply:	5V



CAN Spezifische Hardware:

2x CAN DB9 Steckverbinder

→ D-Sub Stecker 9-Polig (D Sub = Steckersystem)

2x CAN BUS Transceiver („SN65HVD230“, Fa. Texas Instruments)

Konfiguration der Software

„Wie setze ich einen CAN Controller in Betrieb“

- Konfiguration der Pins
  - → Port Pin PD0 & PD1 für CAN Port1 / PB5 und PB6 CAN Port2

```
// <h> Port 0
//   <o0.0..1> P0.0
//     <0=> GPIO
//     <1=> RD1
//     <2=> TXD3
//     <3=> SDA1
//   <o2.0..1> P0.0 Pull Up/Down
//     <0=> Pull Up
//     <2=> Neither Nor
//     <3=> Pull Down
//   <o0.2..3> P0.1
//     <0=> GPIO
//     <1=> TD1
//     <2=> RXD3
//     <3=> SCL1
//   <o2.2..3> P0.1 Pull Up/Down
//     <0=> Pull Up
//     <2=> Neither Nor
//     <3=> Pull Down
//   <o0.4..5> P0.2
//     <0=> GPIO
//     <1=> TXD0
//     <2=> AD0.7
//   <o2.4..5> P0.2 Pull Up/Down
//     <0=> Pull Up
//     <2=> Neither Nor
//     <3=> Pull Down
```

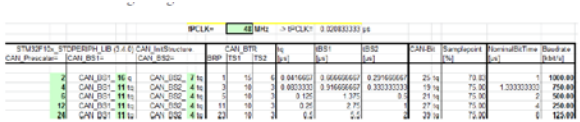
- Konfiguration GPIO´s
  - → Ein-/ Ausgänge deklarieren (gpio.h)

```
  //--Defines-----------------------------------------------------------------
  //
#define _GPIO_F_SET_OUT(PORT,PIN) \
  ((LPC_GPIO_TypeDef*)(LPC_GPIO0_BASE+0x20UL*PORT))->FIODIR |=  (1UL<<PIN)
#define _GPIO_F_SET_IN(PORT,PIN) \
  ((LPC_GPIO_TypeDef*)(LPC_GPIO0_BASE+0x20UL*PORT))->FIODIR &= ~(1UL<<PIN)

#define _GPIO_F_SET(PORT,PIN) \
  ((LPC_GPIO_TypeDef*)(LPC_GPIO0_BASE+0x20UL*PORT))->FIOSET = (1UL<<PIN)
#define _GPIO_F_CLR(PORT,PIN) \
  ((LPC_GPIO_TypeDef*)(LPC_GPIO0_BASE+0x20UL*PORT))->FIOCLR = (1UL<<PIN)

#define _GPIO_F_GET(PORT,PIN_) \
  (((LPC_GPIO_TypeDef*)(LPC_GPIO0_BASE+0x20UL*PORT))->FIOPIN&(1UL<<PIN_))

#define RISING(PORT) IO##PORT##_INT_EN_R
#define FALLING(PORT) IO##PORT##_INT_EN_F
#define _GPIO_ENABLE_INT(PORT,PIN,EDGE) EDGE(PORT) |= (1UL<<PIN)
#define _GPIO_DISABLE_INT(PORT,PIN,EDGE) EDGE(PORT) |= (1UL<<PIN)
```

Konfiguration der Software

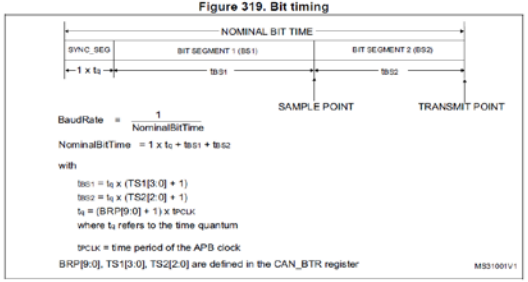
- Variable für die Bitrate definieren
  - → Die Bitrate des CAN Busses ergibt sich aus der Zeit für ein Bit
    Die Bit Zeit bestimmen Prescaler, T segment1 und T segment2

Im Beispiel: Datenrate 250kBit/s



Hier sind exemplarisch für eine f_PCLK = 48 MHz die nötigen Einstellungen gezeigt.

Figure 319. Bit timing



Variable der Bitrate

```
Defines-------------------------------------------
-----------
  //
    //! Applied clock to the can controllers
    #define CAN_PCLK 25000000
```

Berechnung der Bitrate

```
//-----------------------------------------------------------------------------
//! \brief  Applies the specified datarate to the specified can port.
//!
//! \param  pCanPort Pointer to can port to change the datarate.
//! \param  pDataRate New data Rate which is applied.
//
  void canSetDataRate( const tCanPort *pCanPort, const tCanDataRates
    pDataRate)
  {
    uint32_t lValue;
    uint32_t lNominalTime;

    // Determine which nominal time to use for PCLK and baudrate
    if (((CAN_PCLK / 1000000) % 6) == 0)
    {
      lNominalTime = 12;
    }
    else
    {
      lNominalTime = 10;
    }
```

Konfiguration der Software

- Handling von Nachrichten
  - → Senden/Empfangen von Nachrichten
  - → Nachrichtenfilterung
  - → Mask und ID-Listenmodus
  - → Interrupt gesteuerter Nachrichtenempfang

- Abspeichern der Variablen im Register
  - → Ende der Konfiguration

Vielen Dank für Ihre Aufmerksamkeit