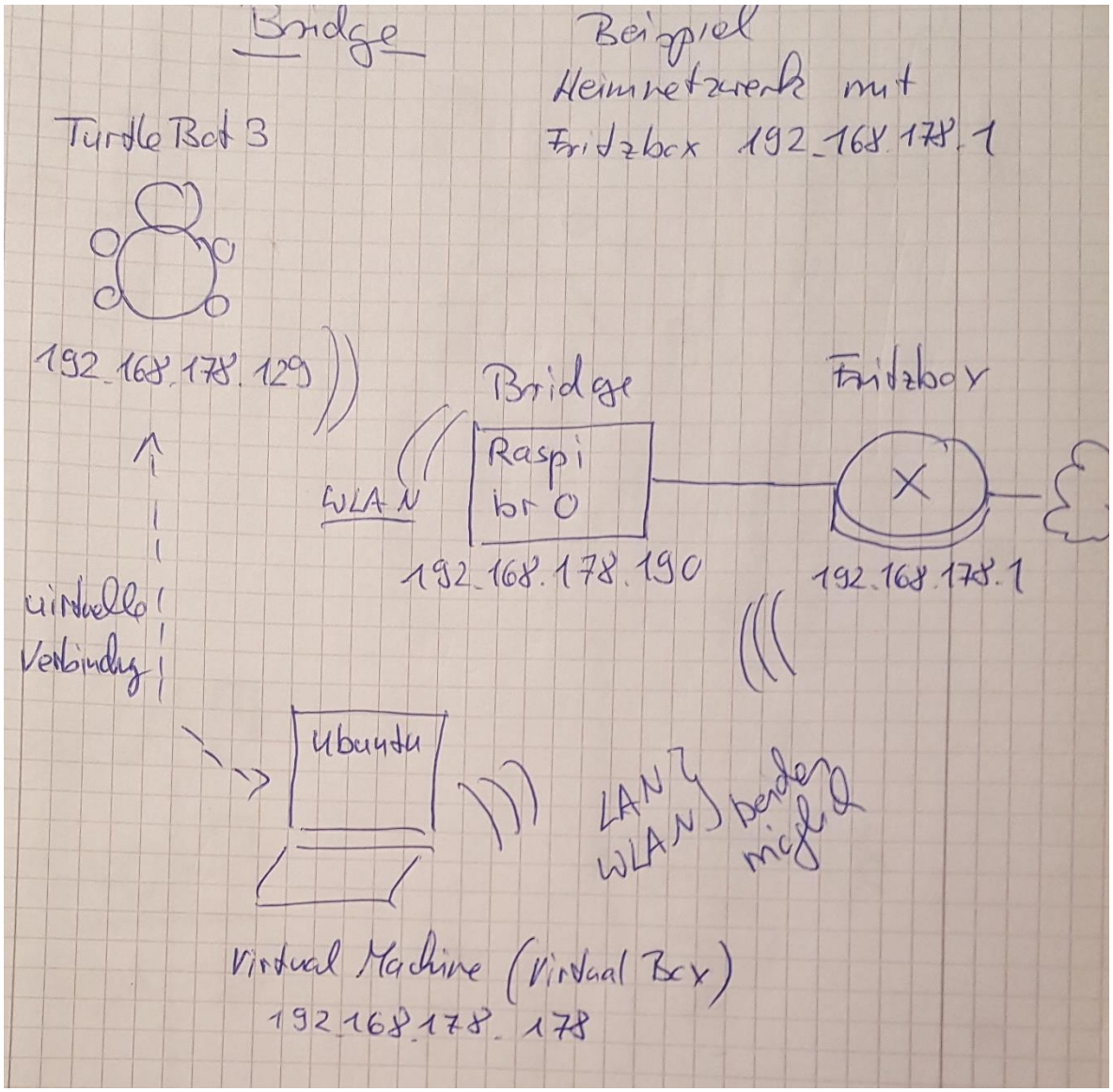
Inhaltsverzeichnis

1 Beispiel einer typischen Datenkommunikation



Die oben gezeigte Skizze zeigt ein typisches Bild für die Kommunikation mehrerer Komponenten.

- TurtleBot: Hier „werkeln“ zwei Mikrocontroller, der Raspi dient der Kommunikation und das OpenCR Board der Steuerung von Motoren und Sensorerfassung
- Bridge: Über diesen Raspi erfolgt die Kommunikation mit dem Router. Dies ist im Heimnetz nicht nötig, allerding im Labor D204, da ansonsten kein Zugang zum Internet möglich ist (geht an der Hochschule nur mit eduroam bzw. vpn.)

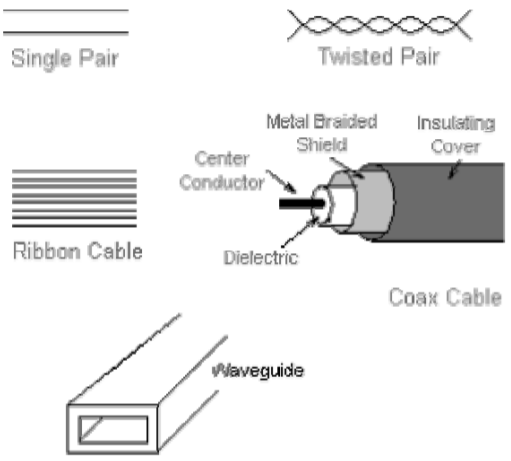
2 Verbindungsleitungen

2.1 Verbindungsleitungen

Für die Wahl der Verbindungsleitungen kann die folgende Tabelle eine ungefähre Richtschnur geben:

Übertragungsrate	Entfernung	Busorganisation	Leitung
Kleiner als 1 Mbit/s	Kleiner als 10 Meter	parallel oder seriell	Geätzte Leiterbahnen, einfacher Draht oder Rund- bzw. Flachbandkabel mit einer Masse
	Größer als 10 Meter	Seriell	Verdrillte oder abgeschirmte Leitung oder Koaxialkabel
Grösser als 1 Mbit/s	Kleiner als 10 Meter	parallel oder seriell	Geätzte Leiterbahnen mit spezieller Masseführung, Flachbandkabel mit Mehrfachmasse, Rundkabel oder Koaxialkabel
	Größer als 10 Meter	Seriell	Verdrillte und abgeschirmte Leitungen, Koaxialkabel oder Lichtwellenleiter

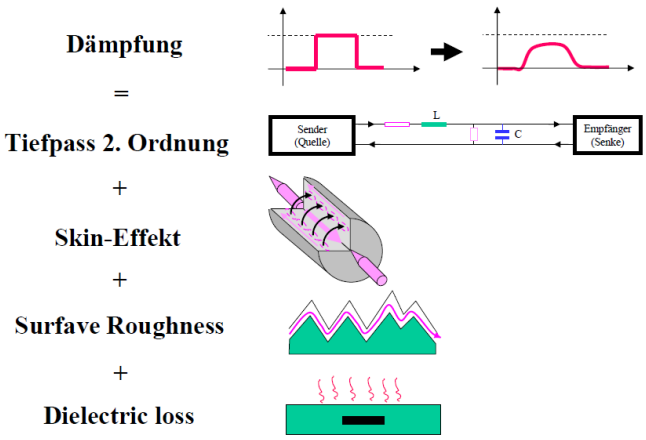
Beispiele für Kabel:



- Das „Single Pair“ kann sich Jeder vorstellen: eine Hinleitung und eine Rückleitung, allerdings exakt in Kunststoff parallel geführt.
- Das „Twisted Pair“ ist die Standardform bei modernen LANs, also Computernetzen (...und z. B. beim CANBus).
- Das Koaxkabel sehen wir als Antennenzuleitung bei jedem Fernsehempfänger.
- „Ribbon Cable“ ist ein vieladriges rundes Kabel oder ein vieladriges Flachbandkabel (z. B. in PCs zur Verbindung der Festplatte oder des CD-Laufwerks mit dem Mainboard).
- Richtig geheimnisvoll wird es erst beim „waveguide“ (= Hohlleiter), denn dort werden die elektrischen und magnetischen Felder nicht zwischen Drähten, sondern in einem „Hohlraum“, also in Luft, geführt.

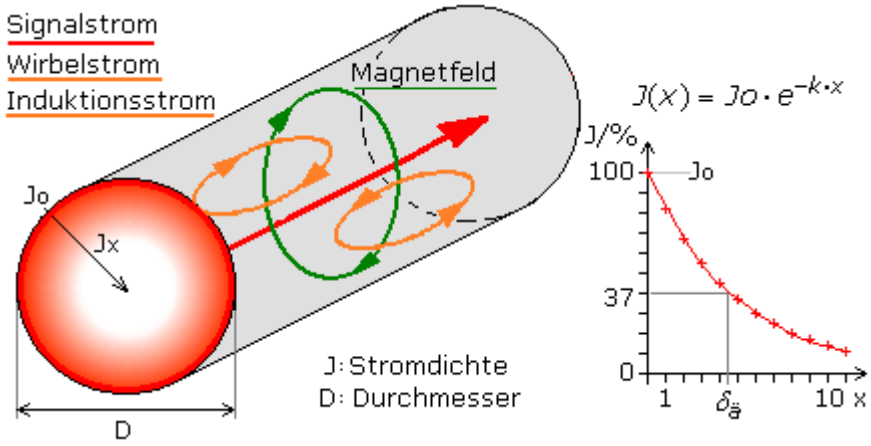
2.2 Leitungsdämpfung

Ein Rechtecksignal wird aufgrund mehrerer Effekte verschliffen. Je nach der Betriebsfrequenz überwiegen unterschiedliche Effekte. Beispielsweise führt der Skineffekt bei hohen Frequenzen dazu dass der Widerstand der Leitung zunimmt, da lediglich die Oberfläche der Leitung den Strom leitet.



2.2.1 Skineffekt[1]

Fließt Wechselstrom durch einen Leiter, so entstehen magnetische Wechselfelder. Sie erzeugen im Leiter nach dem Induktionsgesetz Wirbelströme, die den Erregerstrom zur Leitermitte hin schwächen. In den Randbereichen verlaufen die Erreger- und Wirbelstrompfade eher in der gleichen Richtung. Von der Leiteroberfläche ausgehend nimmt die Stromdichte vom Maximalwert 100% oder 1 zum Leiterinneren hin nach einer e-Funktion ab. Das verringert den wirksamen Leiterquerschnitt und der Widerstand wird mit zunehmender Signalfrequenz größer. In der Skizze ist zur besseren Übersicht nur je eine Magnetfeldlinie und Wirbelstromschleife dargestellt.



Im Exponenten k stehen die Kreisfrequenz des Signals und mit der Leitfähigkeit und der Permeabilität die charakteristischen Materialeigenschaften des Leiters. Je höher die Signalfrequenz ist, desto geringer ist die Stromdichte im Leiterinneren. Der Strom fließt dann nur noch in einem schmalen Bereich nahe der Leiteroberfläche, der Leiterhaut,

[1] https://www.elektroniktutor.de/elektrophysik/leitung.html

englisch: skin, während der Innenraum fast stromfrei bleibt. Der Einfluss des Skineffekts ist im NF-Bereich vernachlässigbar gering und gewinnt erst im HF-Bereich an Bedeutung.

$$J(x) = Jo \cdot e^{-k \cdot x} = Jo \cdot e^{-\sqrt{\frac{2}{\omega \kappa \mu}} \cdot x}$$

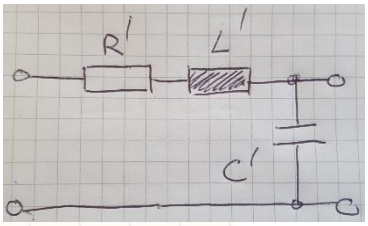
$$\frac{J(x)}{Jo} = e^{-\sqrt{\frac{2}{\omega \kappa \mu}} \cdot x} = e^{-1} \quad \text{für} \quad x = \delta_{\ddot{a}}$$

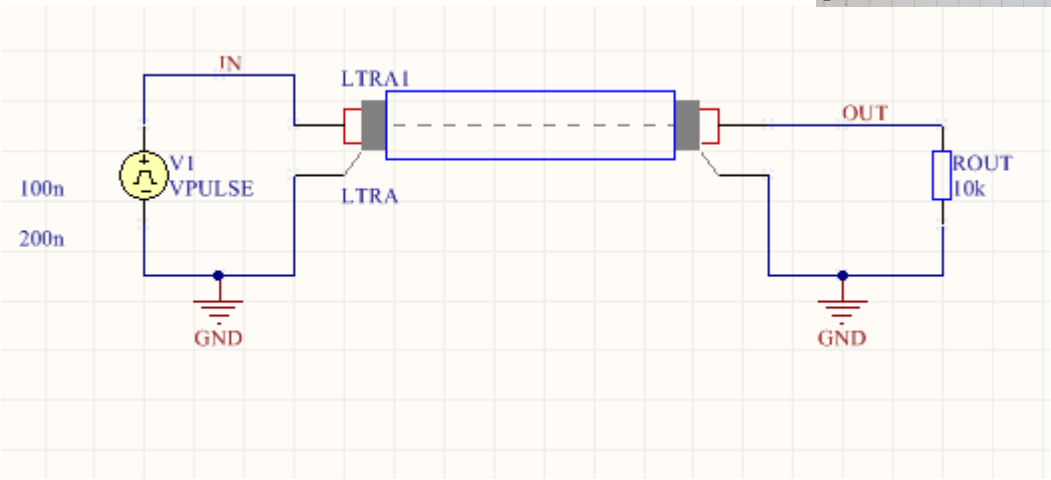
$$\delta_{\ddot{a}} = \frac{1}{k} = \sqrt{\frac{\omega \mu \kappa}{2}} = \frac{1}{\sqrt{\pi f \mu_0 \mu_r \kappa}}$$

2.3 Beispiel für Leitungsabschuss

Das folgende Beispiel zeigt exemplarisch den Einfluss von Abschlusswiderständen auf den Signalverlauf. Die Simulation wurde mit Spice durchgeführt (Altium Designer).
Die Leitungsbeläge eines verdrillten Flachbandkabels haben laut Hersteller die folgenden Werte:

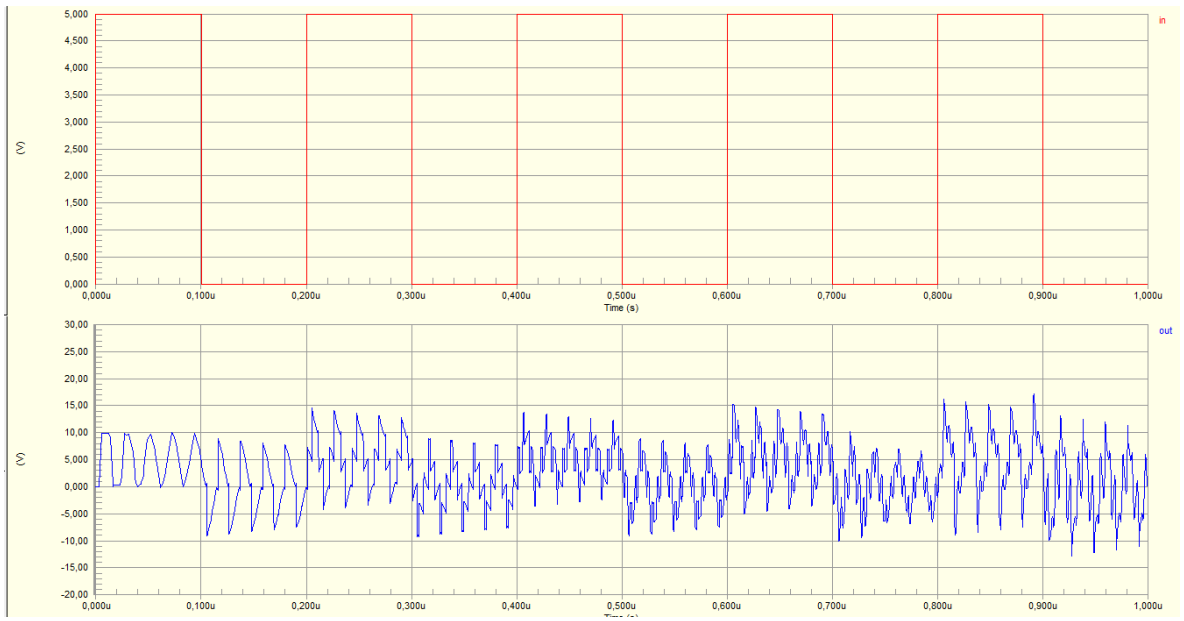
- R´= 0.2 Ohm/m, L´ = 600nH/m und C´= 47pF/m.
- Die Länge der Leitung ist 1m.
- Der Wellenwiderstand der Leitung ist 115 Ohm.





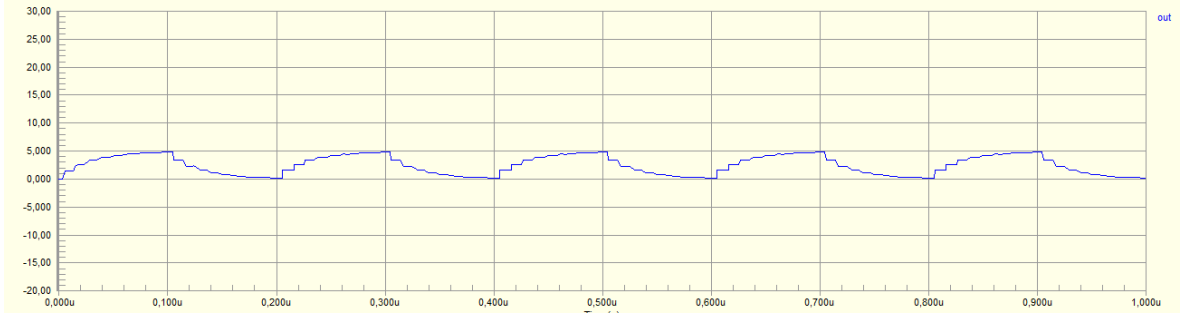
Das Signal hat eine Frequenz von 5 MHz und 50 % Dutycycle, eine Rechteckspannung mit keiner besonders hohen Frequenz. Die folgenden Bilder zeigen den Effekt für eine fehlende Anpassung an den Wellenwiderstand des Kabels.

ROUT = 10kOhm



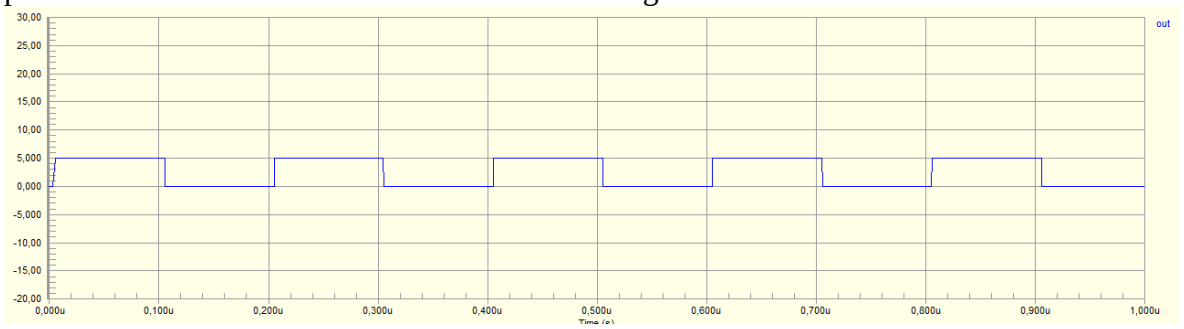
Bei einer fast offenen Leitung ist das Signal durch Reflexionen praktisch unbrauchbar

ROUT = 20Ohm



Durch die fast kurzgeschlossene Leitung wird das Signal stark tiefpassgefiltert.

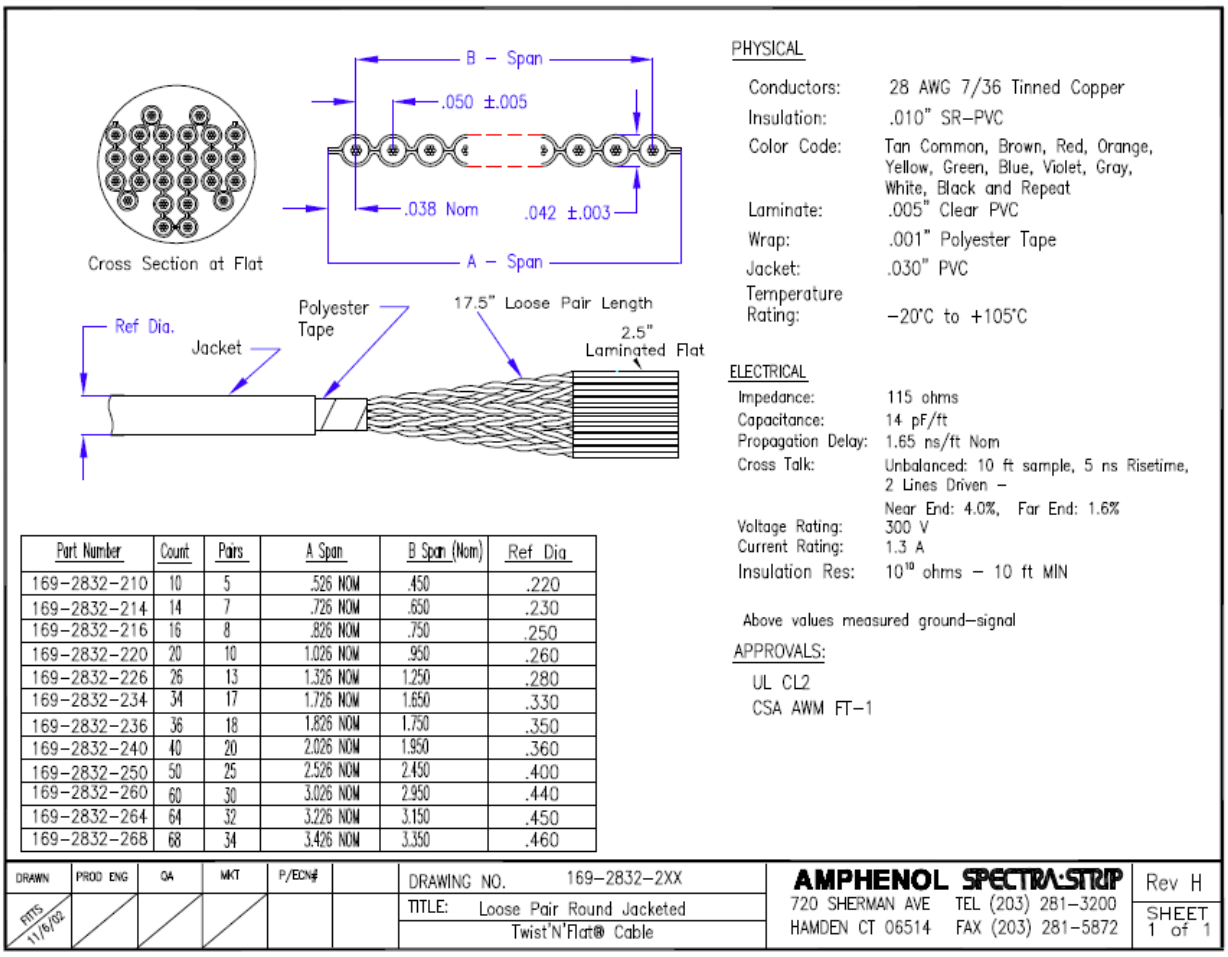
Bei einer korrekten Anpassung an das Kabel mit ROUT=115Ohm ist das Signal nahezu perfekt bis auf die unvermeidliche Laufzeit des Signals.



Fragen:

Wie sieht das Ersatzschaltbild einer verlustbehafteten Leitung aus?
Was versteht man unter dem Begriff Wellenwiderstand?
Warum ist das Ausgangssignal gegenüber dem Eingangssignal zeitlich verschoben, auch bei idealer Anpassung an den Wellenwiderstand?

Datenblatt eines typischen Flachband/Rundkabel:



Part Number	Count	Pairs	A Span	B Span (Nom)	Ref Dia
169–2832–210	10	5	.526 NOM	.450	.220
169–2832–214	14	7	.726 NOM	.650	.230
169–2832–216	16	8	.826 NOM	.750	.250
169–2832–220	20	10	1.026 NOM	.950	.260
169–2832–226	26	13	1.326 NOM	1.250	.280
169–2832–234	34	17	1.726 NOM	1.650	.330
169–2832–236	36	18	1.826 NOM	1.750	.350
169–2832–240	40	20	2.026 NOM	1.950	.360
169–2832–250	50	25	2.526 NOM	2.450	.400
169–2832–260	60	30	3.026 NOM	2.950	.440
169–2832–264	64	32	3.226 NOM	3.150	.450
169–2832–268	68	34	3.426 NOM	3.350	.460

2.3.1 Kategorisierung von Rundkabel

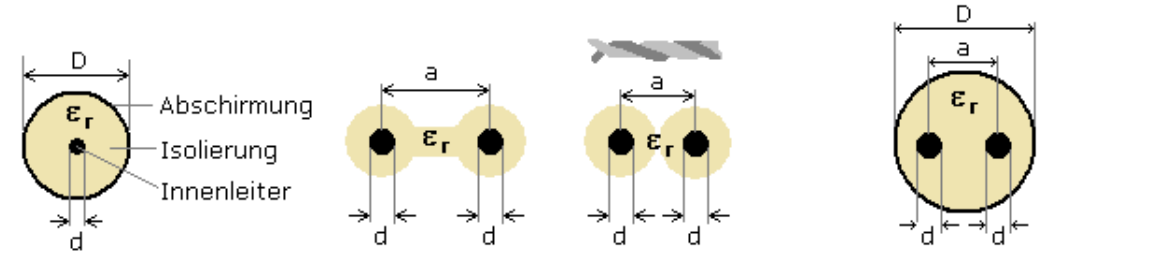
Kategorie	Übertragungsgeschwindigkeit	Frequenz	Max. Kabellänge
Cat3	10 MBit/s	16 MHz	100 m
Cat5	100 MBit/s	100 MHz	100 m
Cat5e	1 Gigabit/s	100 MHz	100 m
FastCat5e	1 Gigiabit/s	350 MHz	100 m
Cat6	1 Gigabit/s	250 MHz	100 m
Cat6a	10 Gigabit/s	500 MHz	100 m
Cat7	10 Gigabit/s	600 MHz	100 m
Cat7a	100 Gigabit/s	1.000 MHz	100 m

Es gibt dann zu jeder Kategorie unterschiedliche Varianten z.B. UTP (Unshielded Twisted Pair) und STP (Shielded Twisted Pair).

Die Diskrepanz zwischen Übertragungsgeschwindigkeit und Frequenz kommt durch die eingesetzten Übertragungsverfahren zustande

2.4 Wellenwiderstand

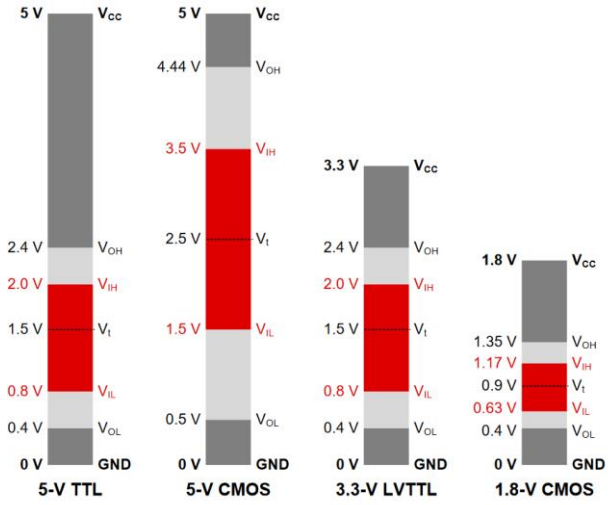
Die folgenden Bilder zeigen die Berechnung des Wellenwiderstands unterschiedlicher Leitungen[2]



$$Z = \frac{60}{\sqrt{\varepsilon_r}} \cdot \ln\frac{D}{d} \qquad Z = \frac{120}{\sqrt{\varepsilon_r}} \cdot \ln\frac{2a}{d} \qquad Z = \frac{120}{\sqrt{\varepsilon_{r\,eff}}} \cdot \ln\frac{2a}{d} \qquad Z = \frac{120}{\sqrt{\varepsilon_r}} \cdot \ln\left(\frac{2a}{d} \cdot \frac{D^2 - a^2}{D^2 + a^2}\right)$$

[2] https://www.elektroniktutor.de/elektrophysik/leitung.html

3 Spannungspegel auf Mikrocontrollerseite



Da es unterschiedliche Spannungspegel auf der Mikrocontrollerseite gibt, müssen Spannungen zwischen den angeschlossenen Komponenten in der Regel angepasst werden.

Das folgende Bild zeigt exemplarisch die Wandlung von 1,8V auf 3,3V Logik.

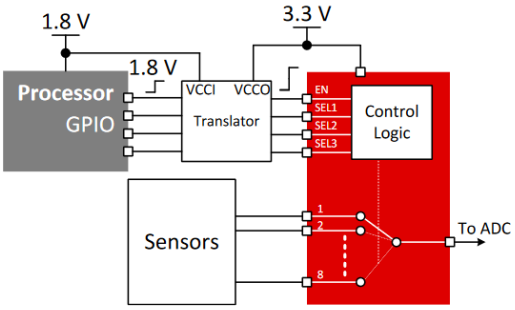
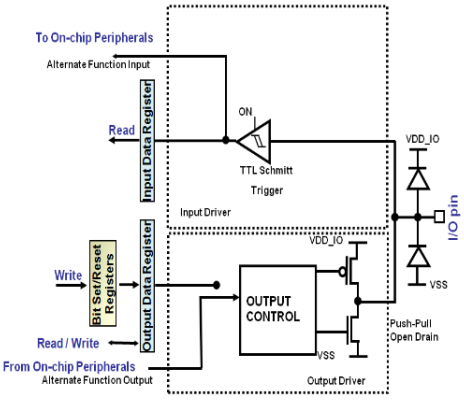


Figure 3. 8:1 MUX without 1.8 V Logic

3.1 Aufbau eines typischen GPIO

Ein typischer GPIO Pin hat den neben gezeigten prinzipiellen Aufbau. Er kann je nach Bedarf konfiguriert werden. Die wichtigsten Einstellungen sind Push Pull, Open Drain und Analog.

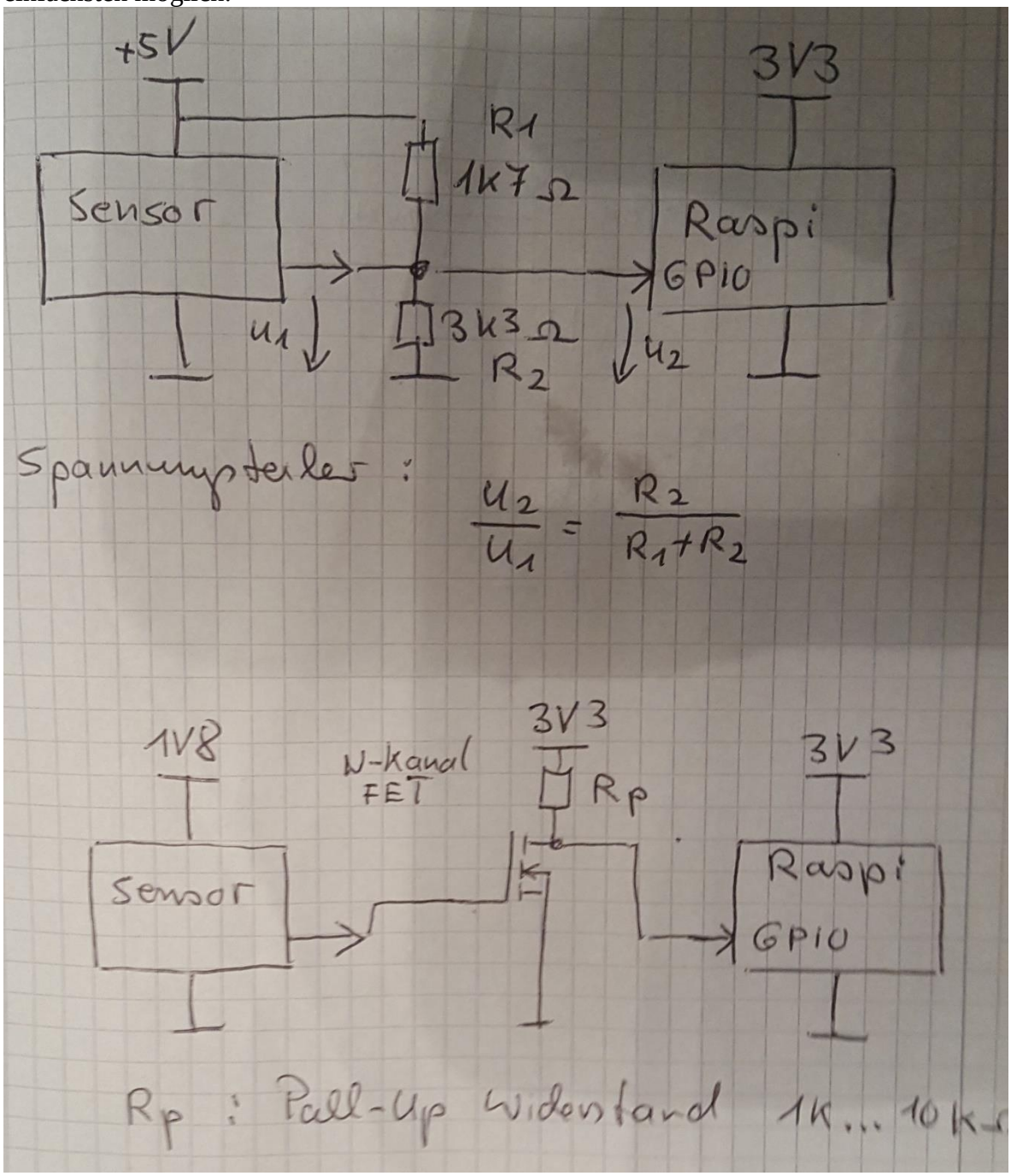
Configuration Mode	CNF1	CNF0	MOD1	MOD0
Analog Input	0	0	00	
Input Floating (Reset State)	0	1		
Input Pull-Up	1	0		
Input Pull-Down	1	0		
Output Push-Pull	0	0	00: Reserved 01: 10 MHz 10: 2 MHz 11: 50 MHz	
Output Open-Drain	0	1		
AF Push-Pull	1	0		
AF Open-Drain	1	1		



GPIO STM32F4 Discovery 1-7

3.1.1 Einfache Pegelanpassung

Unidirektional von höherer zu niedriger Spannung ist über einen Spannungsteiler am einfachsten möglich:

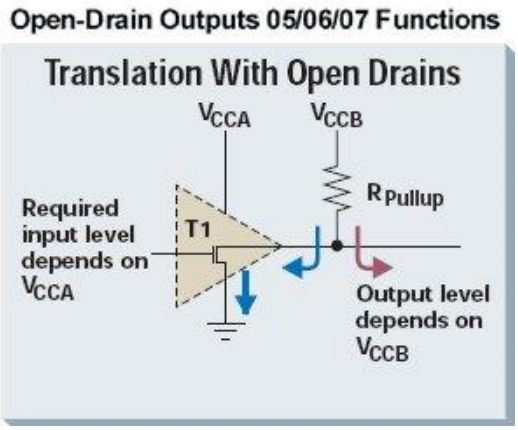


Von einer geringeren zu einer höheren Spannung ist mit einer open Drainschaltung möglich.

Falls mehrere Kanäle umgesetzt werden sollen bietet sich eine integrierte Treiberschaltung an.[3]

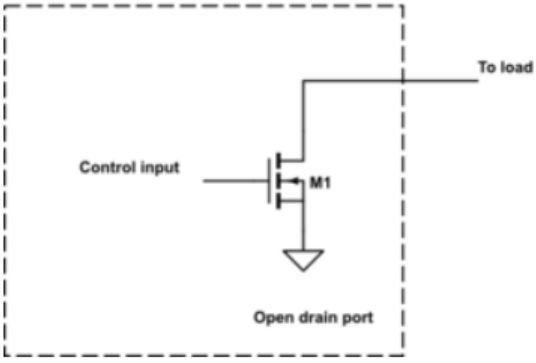
[3] https://www.mikrocontroller.net/articles/Pegelwandler

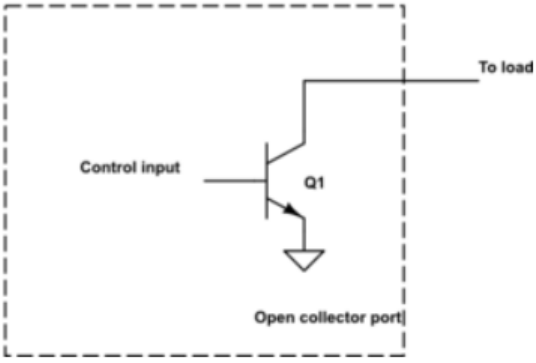
3.1.2 Open Drain Ausgang



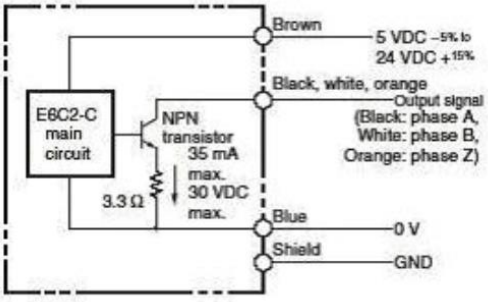
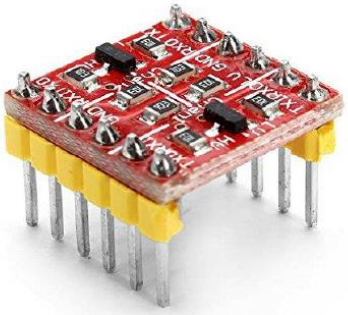
Open Drain Ausgänge werden für Pegelanpassungen verwendet. Das Bild zeigt die Anpassung des Ausgangs des ICs mit Vcca an einen Verbraucher mit der Spannung Vccb.

Eine weitere sehr wichtige Eigenschafdt von Open Drain ist die Möglickeit von Boolschen Verknüpfungen. Wenn mehrere Transistoren als Schalter betrachtet werden und mit einem gemeinsamen Widerstand parallel gegen Vcc verbunden ergibt sich eine Wired NOR Verknüpfung. Diese Eigenschaft wird z.B. bei dem seriellen I2C-Bus genutzt.
Links ist ein Open-Drain Ausgang gezeichnet und rechts ein Open Collector Ausgang.

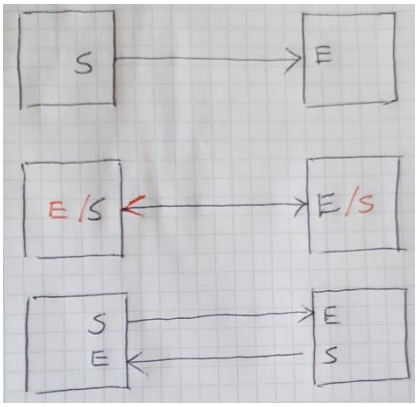




Typische Open Collector Ausgangsstufe:

Schaltung E6C2-CWZ6C (NPN open collector output)	Pegelwandler 3.3V – 5.0 Volt
	

4 Simplex, Halb und Vollduplex



Simplexübertragung

Halbduplexübertragung

Vollduplxübertragung

5 UART (mit RS232 und RS485)

Ausschnitt aus dem Referenzmanual des STM32F072RB Mikrocontrollers:

27.2 USART main features

• Full-duplex asynchronous communications
• NRZ standard format (mark/space)
• Configurable oversampling method by 16 or 8 to give flexibility between speed and clock tolerance
• A common programmable transmit and receive baud rate of up to 6 Mbit/s when the clock frequency is 48MHz and oversampling is by 8
• Dual clock domain allowing: – USART functionality and wakeup from Stop mode – Convenient baud rate programming independent from the PCLK reprogramming
• Auto baud rate detection
• Programmable data word length (7, 8 or 9 bits)
• Programmable data order with MSB-first or LSB-first shifting
• Configurable stop bits (1 or 2 stop bits)
• Synchronous mode and clock output for synchronous communications
• Single-wire Half-duplex communications
• Continuous communications using DMA
• Received/transmitted bytes are buffered in reserved SRAM using centralized DMA
• Separate enable bits for transmitter and receiver
• Separate signal polarity control for transmission and reception
• Swappable Tx/Rx pin configuration
• Hardware flow control for modem and RS-485 transceiver

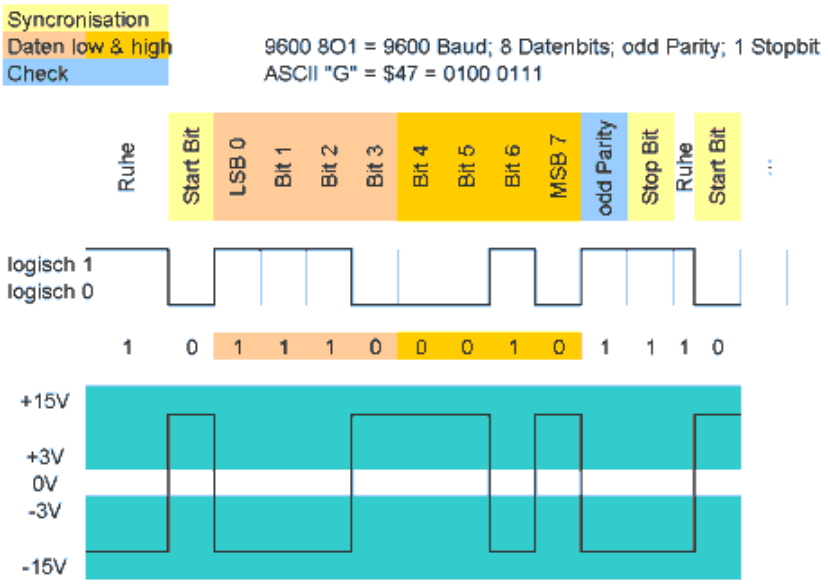
In der Frühzeit der PCs waren für die UART externe ICs eingesetzt (z.B. NSC82509 bei den IBM PCs in den 80er Jahren.
Die oben gezeigte Aufzählung der wichtigsten Eigenschaften der UART am Beispiel eines STM Mikrocontrollers zeigt die Flexibilität dieser Schnittstelle.

Einige Anmerkungen aus der Praxis, einige bereitgestellte Eigenschaften der UART werden eher sehr selten verwendet:

- Die Verwendung als USART, also als eine serielle synchrone Schnittstelle ist weitgehend veraltet
- Hardware Flow Control ist in der Praxis weitgehend verschwunden (RTS und CTS Signale
- In der Regel ist der Rahmen 8-n-1, also 8 Datenbits, keine Parität und 1 Stoppbit.

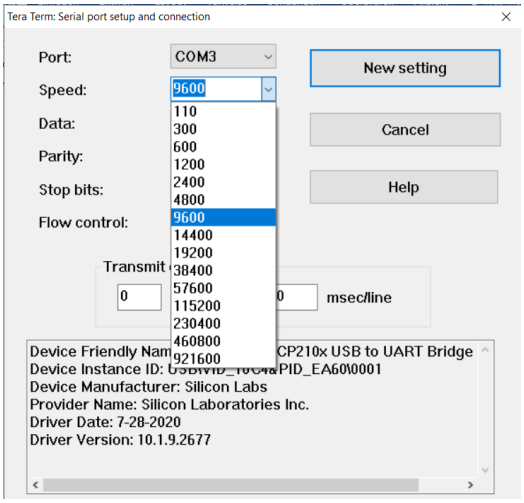
5.1 Rahmen

Das Bild zeigt den Rahmen einer UART Übertragung. Das untere Bild zeigt darüber hinaus die RS232 Spannungspegel



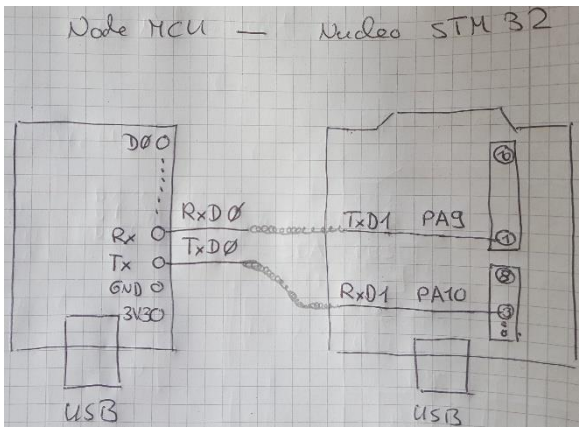
5.2 Datenraten

Es ist möglich Datenraten über 1 Mbit/s zu erreichen, allerdings sind aus der Urzeit der Datenkommunikation die üblichen Datenraten erhalten geblieben. Das Bild zeigt die Einstellungen am Beispiel des Terminalprogramms Teraterm (Freeware). Am meisten werden die Einstellungen 9600 und 115200 Baud verwendet.



5.3 Verbindung zweier UART

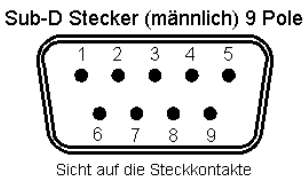
Das Beispiel zeigt eine UART Verbindung eines NodeMCU mit einem STM32 NucleoBoard über die UART1 (UART2 ist bereits über den angeschlossenen Debugger verwendet):

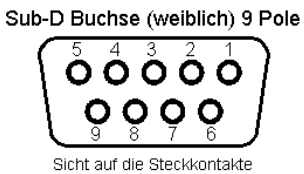


5.4 RS232

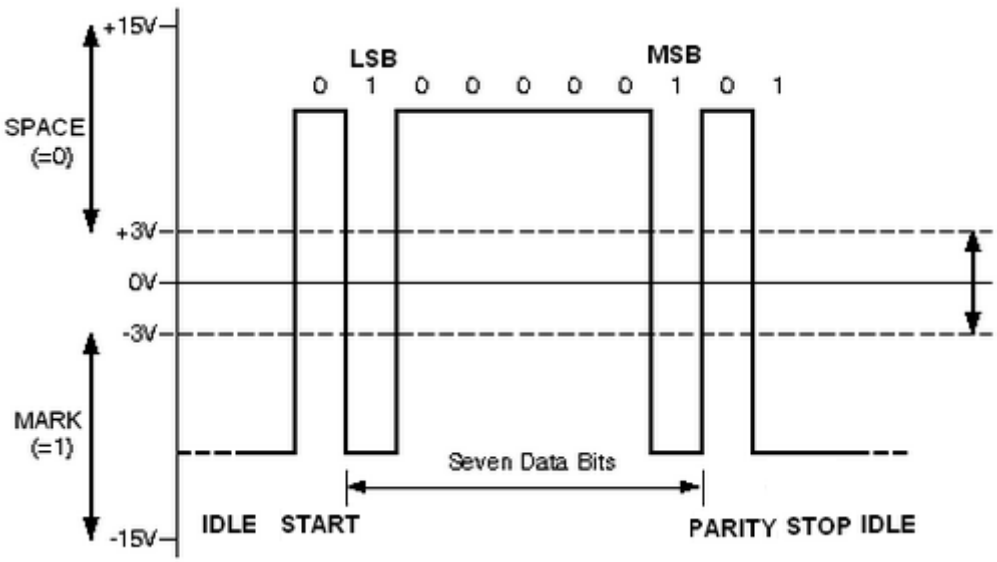
Die RS232 Schnittstelle benutzt die UART als Basis und erweitert diese um Steckverbinder und Spannungspegel.

5.4.1 Steckverbinder

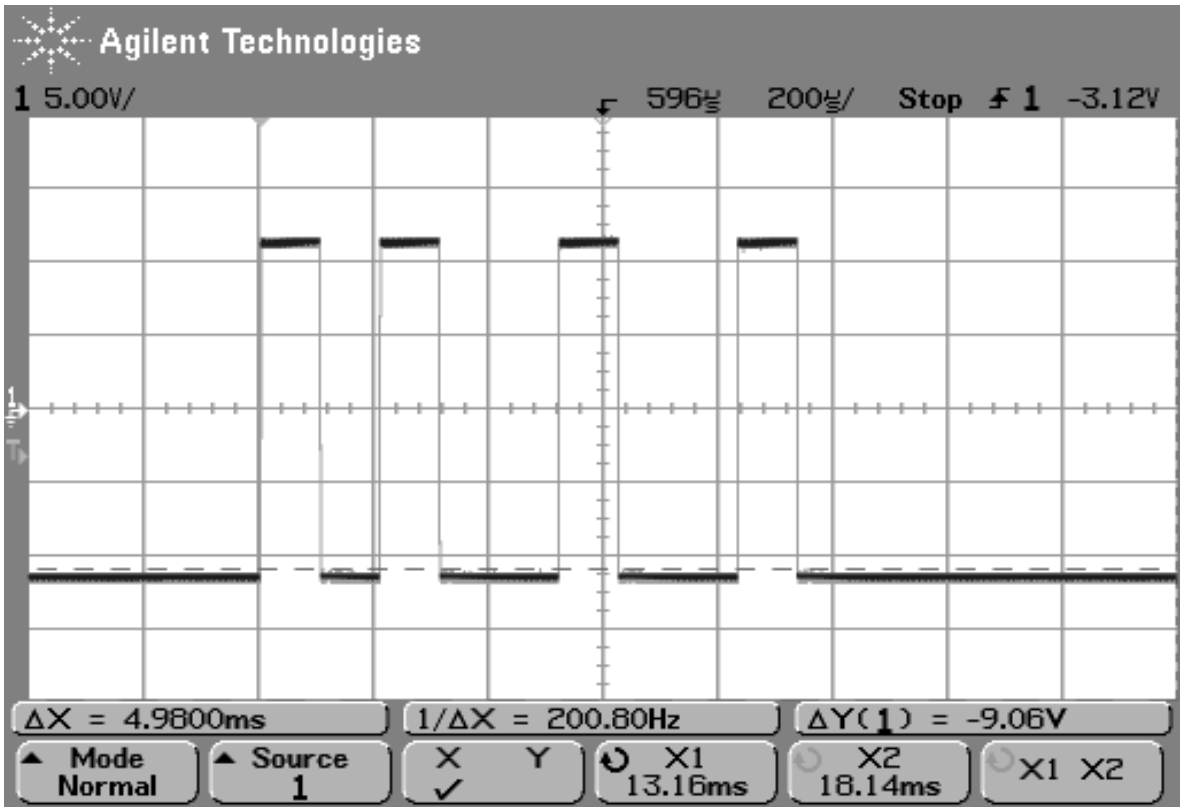




5.4.2 Pegel

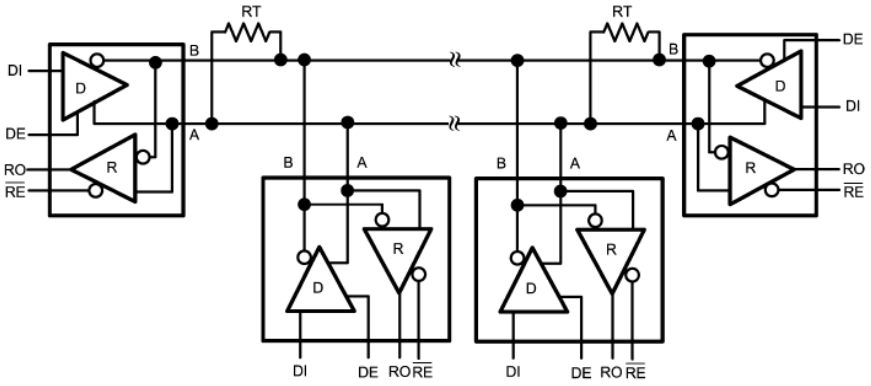


5.4.3 Oszibild RS232



5.5 RS485

Die Basis für die RS485 Schnittstelle ist die UART. Der Unterschied ist hauptsächlich die differentielle Übertragung:



Die Signale A und B sind invertiert zueinander.

Da der Empfänger die Differenz der beiden Signale auswertet, werden Gleichtaktstörungen weitgehend eliminiert.

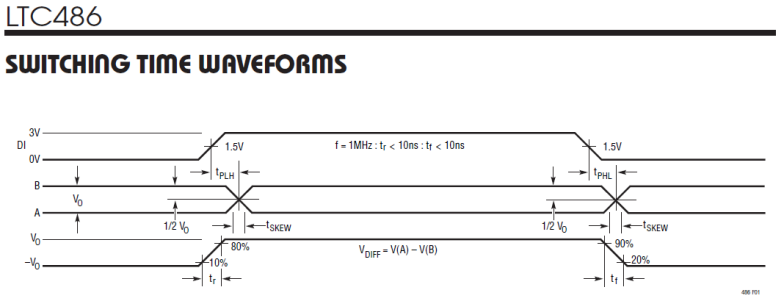


Figure 1. Driver Propagation Delays

5.5.1 *RS485 Transceiver*

Das Bild zeigt einen typischen RS485 Treiber, für den Vollduplexbetrieb sind 4 Leitungen nötig.

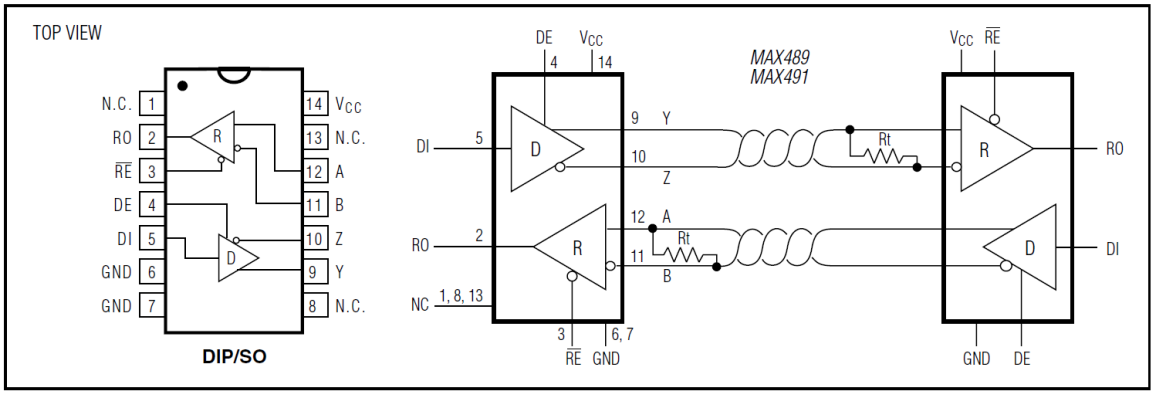
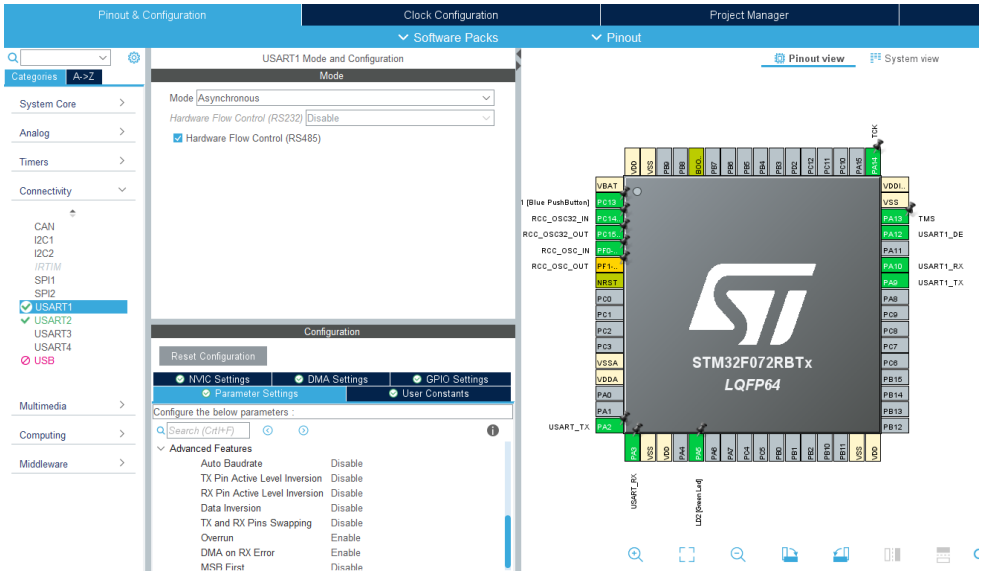


Figure 3. MAX489/MAX491 Pin Configuration and Typical Operating Circuit

5.5.2 RS485 STM3

Das folgende Bild zeigt die Konfiguration der RS485 Schnittstelle mit der CubeIDE für den STM32 Mikrocontroller



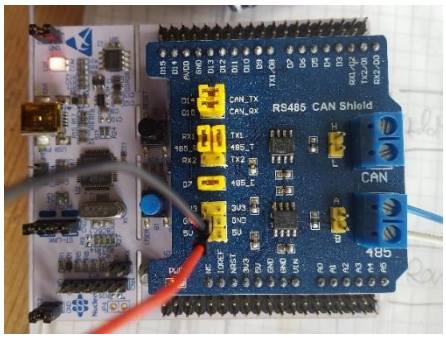
5.5.3 Vergleichstabelle RS232-RS422-RS485

Characteristics of RS232, RS422, RS423 and RS485

	RS232	RS423	RS422	RS485
Differential	no	no	yes	yes
Max number of drivers Max number of receivers	1 1	1 10	1 10	32 32
Modes of operation	half duplex full duplex	half duplex	half duplex	half duplex
Network topology	point-to-point	multidrop	multidrop	multipoint
Max distance (acc. standard)	15 m	1200 m	1200 m	1200 m
Max speed at 12 m Max speed at 1200 m	20 kbs (1 kbs)	100 kbs 1 kbs	10 Mbs 100 kbs	35 Mbs 100 kbs
Max slew rate	30 V/µs	adjustable	n/a	n/a
Receiver input resistance	3..7 kΩ	≧ 4 kΩ	≧ 4 kΩ	≧ 12 kΩ
Driver load impedance	3..7 kΩ	≧ 450 Ω	100 Ω	54 Ω
Receiver input sensitivity	±3 V	±200 mV	±200 mV	±200 mV
Receiver input range	±15 V	±12 V	±10 V	−7..12 V
Max driver output voltage	±25 V	±6 V	±6 V	−7..12 V
Min driver output voltage (with load)	±5 V	±3.6 V	±2.0 V	±1.5 V

5.5.4 Arduino Aufsteckboard mit RS485/CAN

Das folgende Bild zeigt ein Aufsteckmodul (-Shield) für den RS485 und den CAN-Bus. Die Treiber sind unterhalb der Schraubkontakte, jeweils im SOIC Gehäuse:

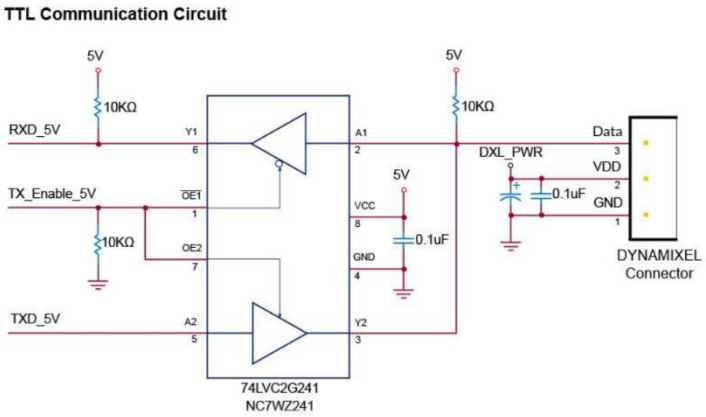


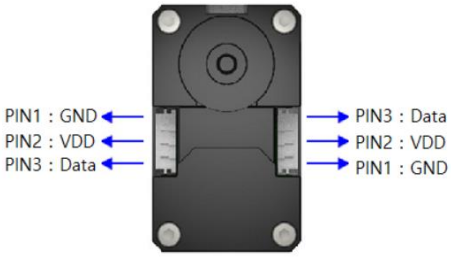
5.5.5 Praktisches Beispiel Dynamixel Servo

Dies sind sehr leistungsfähige Servos mit integriertem ARM Prozessor. Es können bis zu 127 Servos über einen gemeinsamen Bus angesteuert werden.

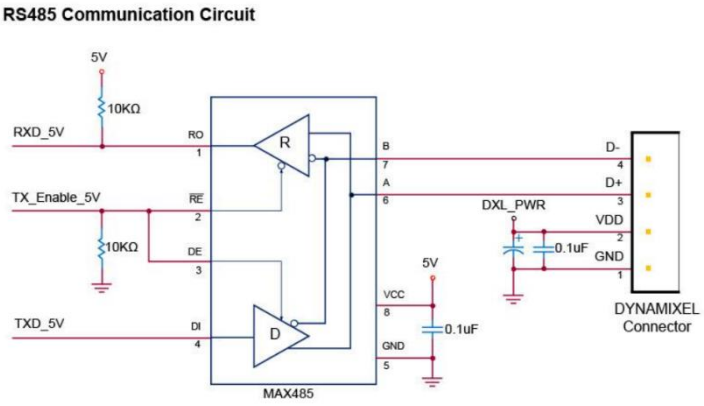
XM430-W350

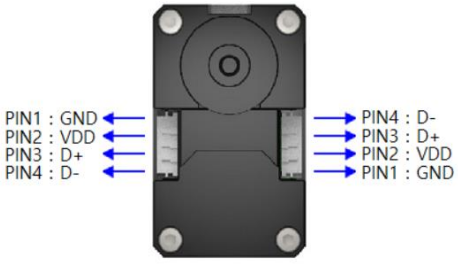
Variante 1: Der Servo wird halbduplex über eine TTL-Spannung gesteuert





Variante 2: Der Servo wird direkt halbduplex mit RS485 Pegeln angesteuert.





6 I2C-Bus

Einige Schlagworte zu dem I2C-Bus:

- Der Bus stammt von Philips aus den 80er Jahren
- Die beiden Leitungen werden mit SDA und SCL bezeichnet
- Es existieren mehrere Bezeichnungen (Patentrechtliche Gründe)
    - TWI (Atmel)
    - SMB (Silicon Labs)
- Es handelt sich um einen Master Slave Bus mit verschiedenen Betriebsarten:
    - Master Transmitter
    - Master Receiver
    - Slave Transmitter
    - Slave Receiver
- Theoretisch ist auch Multimasterbetrieb möglich mit Kollisionerkennung (wird aber in der Praxis so gut wie nie verwendet)
- 8 Bit orientiert
- Protokoll mit 7 Bit Adressierung
- Acknowledge für die Bestätigung
- Maximale Buskapazität 400 pF

6.1 Datenrate

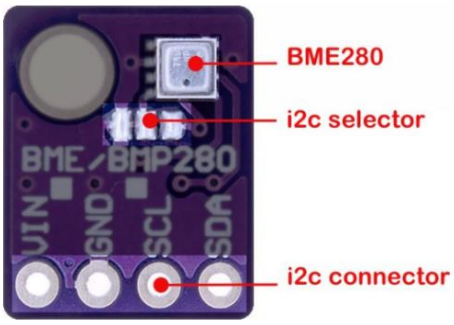
Üblich sind 5 spezifizierte Datenraten:

- Standard Mode ≤ 100 kbit/s
- Fast Mode ≤ 400 kbit/s
- Fast Mode Plus ≤ 1000 kbit/s
- High Speed Mode ≤ 3400 kbit/s
- Ultra Fast Mode ≤ 5000 kbit/s, unidirectional

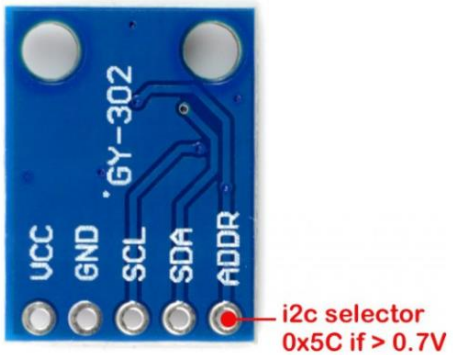
Die maximale Datenrate hängt vom Hersteller ab. Üblich ist eine Datenrate kleiner als 1 Mbit/s, in der Regel aber zwischen 100 – 400 kbit/s.

6.2 Adressierung

Das folgende Bild zeigt zwei Breakoutboards mit einstellbaren Adressen:



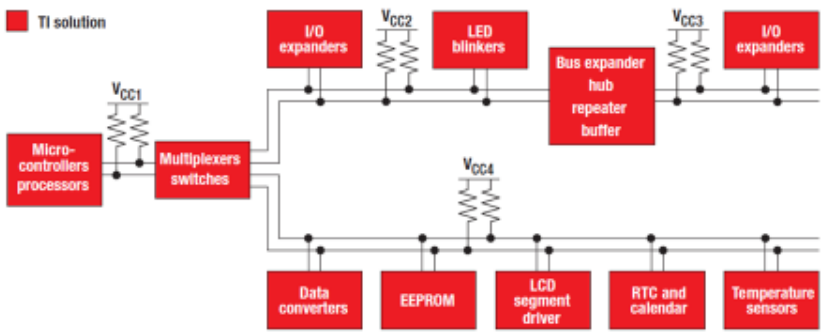
To assign the i2c address 0x77, you have to solder a jumper between the two right pins and cut the track between the two



To assign the address 0x5C, a voltage greater than 0.7V must be applied to the ADDR pin

6.3 Typische I2C ICs

Das folgende Bild zeigt eine Auswahl von ICs, die in einem I2C-Bus häufig eingesetzt werden:



6.4 I2C-Pins

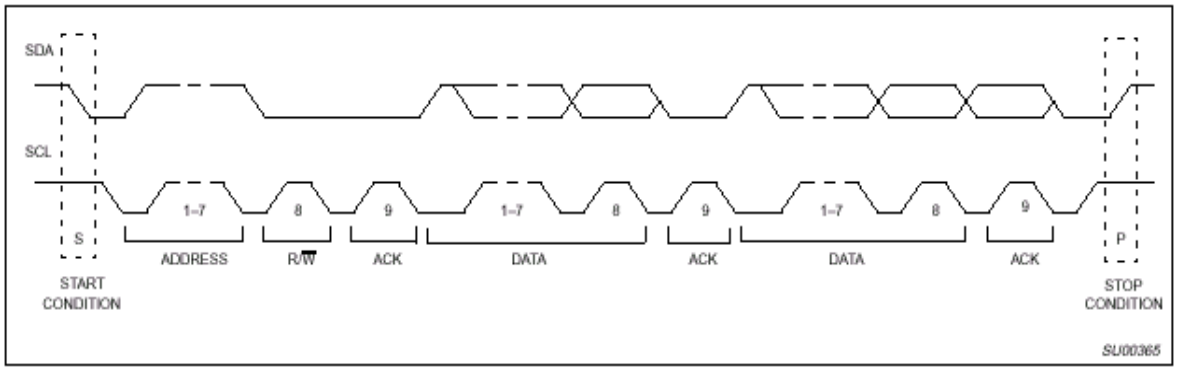
Die allermeisten Mikrocontroller bieten dedizierte Pins für den I2C-Bus an:

Prof. Dr. Parzhuber, Datenkommunikation 16

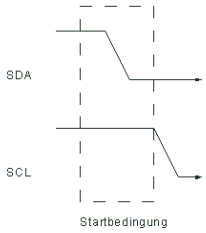
	SDA	SCL	SDA1	SCL1
	Main bus		Secondary bus	
One*	A4	A5		
Ethernet	A4	A5		
Mega2560	20	21		
Leonardo	2	3		
Due	20	21	SDA1	SCL1
ESP8266	D1	D2		
ESP32 DevkitC v4	21	22	User	User
ESP32 (older generations)	IO21	IO22	User	User
STM32	PB7 or PB9	PB6 or PB8	PB11	PB10
Raspberry Pi (any generation)	3	5		

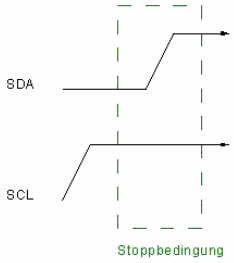
6.5 Protokoll

Das Protokoll ist im folgenden Bild dargestellt:



- Die Übertragung beginnt mit einer Startbedingung und endet mit einer Stoppbedingung





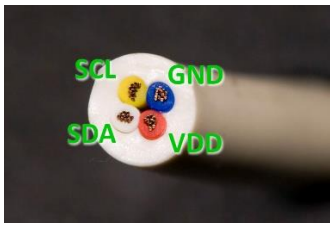
Im Ruhezustand sind die beiden Busleitungen auf High Pegel.

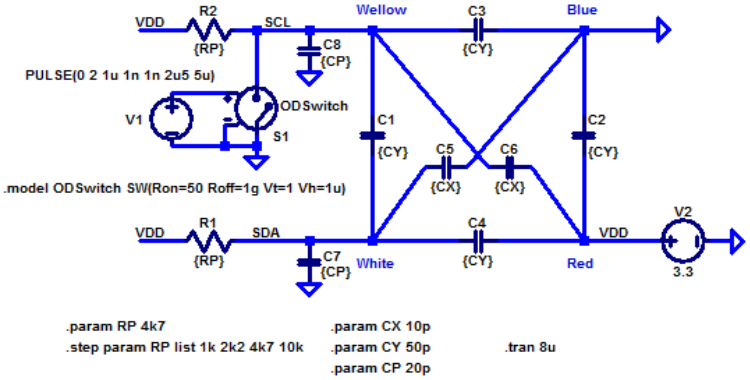
- Die 7-Bitadresse ermöglicht theoretisch 127 Adressen.
- Umschaltung zwischen Schreiben und Lesen erfolgt über das 8te Bit. R/W Bit =0 bedeutet Schreiben und R/W Bit = 1 bedeutet Lesen.
- Wenn das 9te Bit den Wert 0 hat bedeutet dies eine ppsitive Bestätigung des Empfängers der Nachricht.

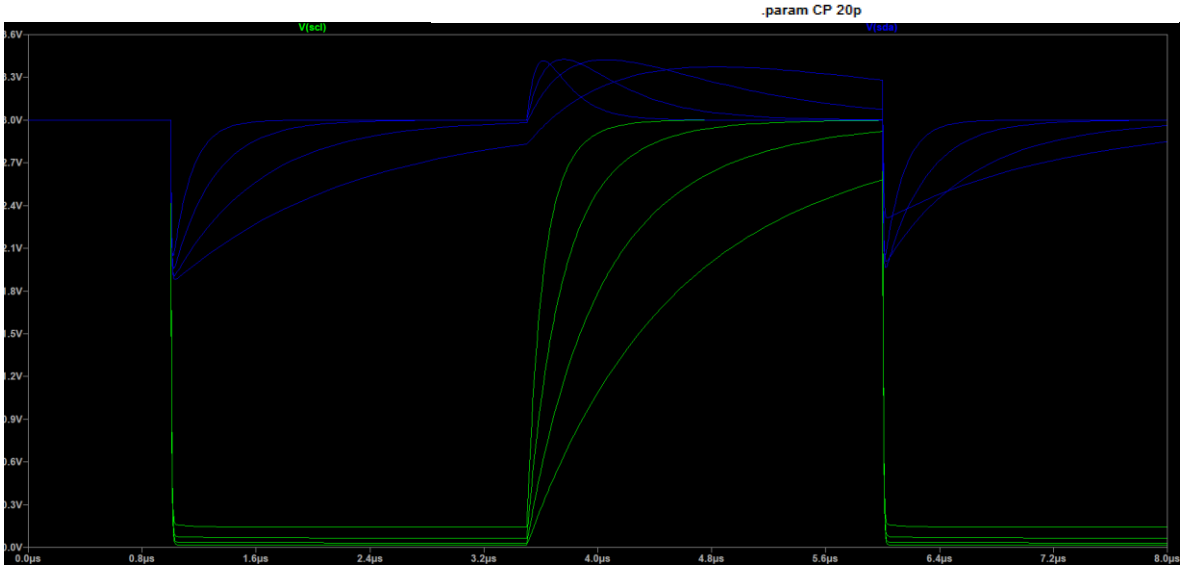
Fragen:

- Wie viele Bits werden für das Senden von zwei Datenbytes an eine beliebige Adresse gesendet?
- Wie lange dauert diese Übertragung bei einer Taktrate von 10 kHz?

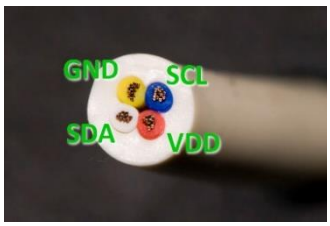
6.6 Simulation mit LTSpice[4]

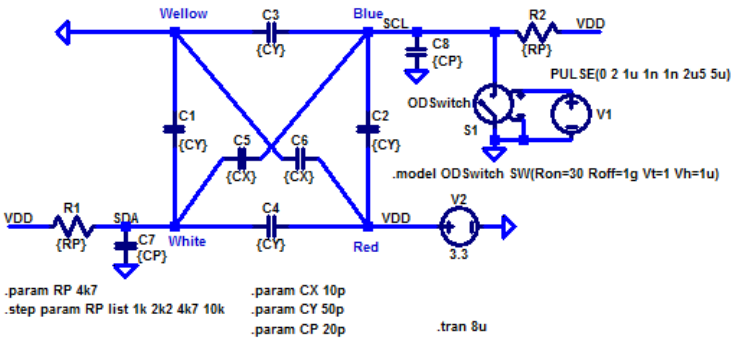




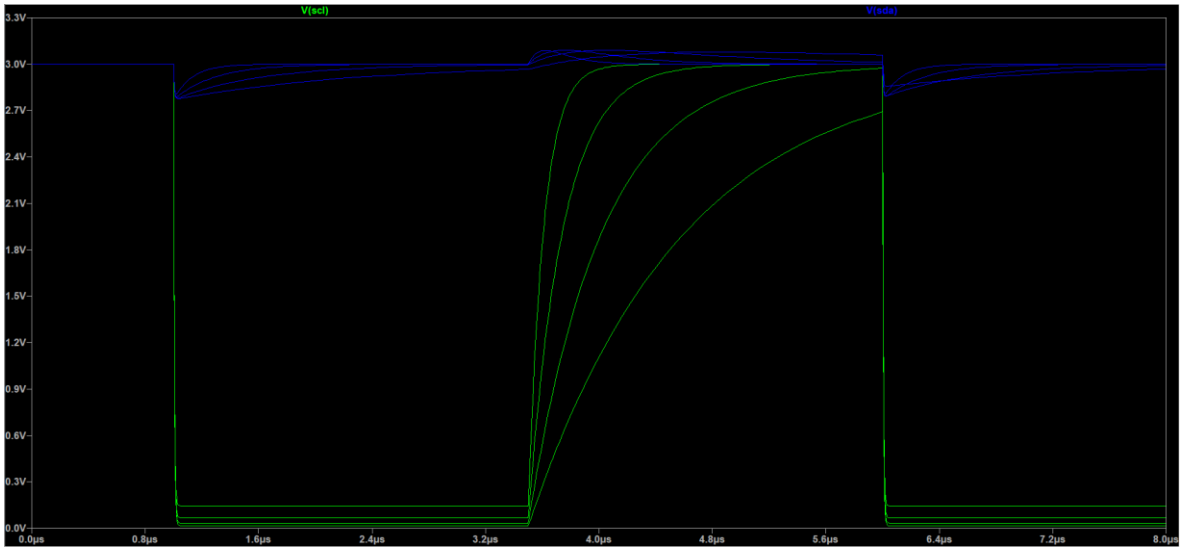


Optimale Anordnung von SCL und SDA im Kabel

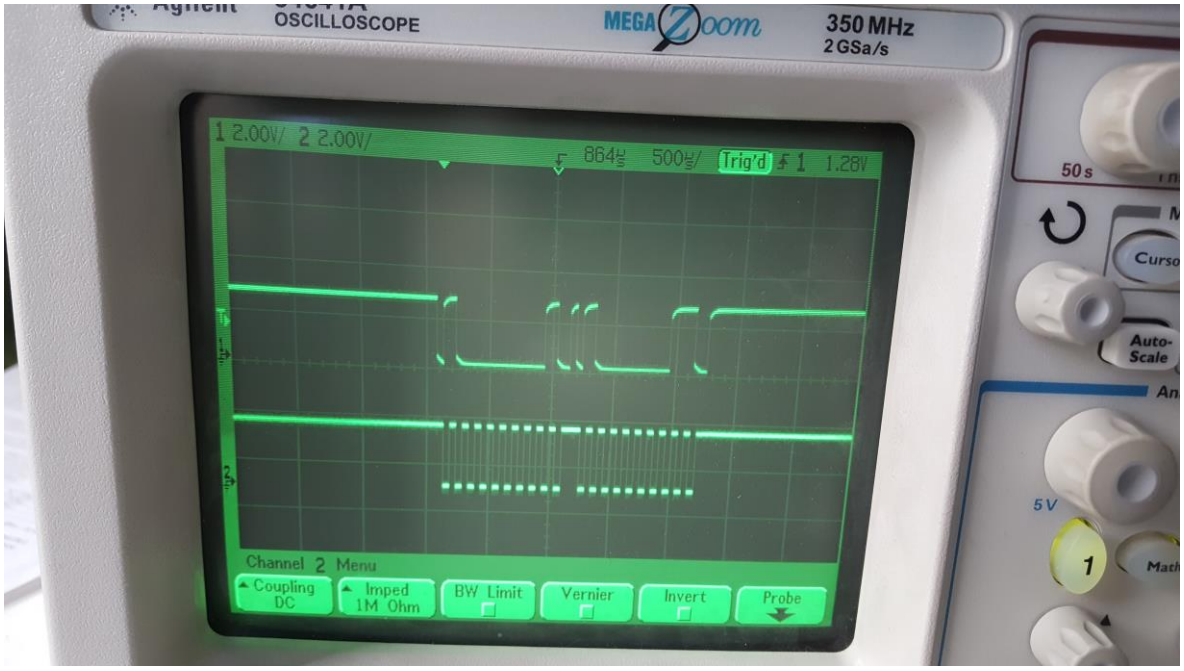




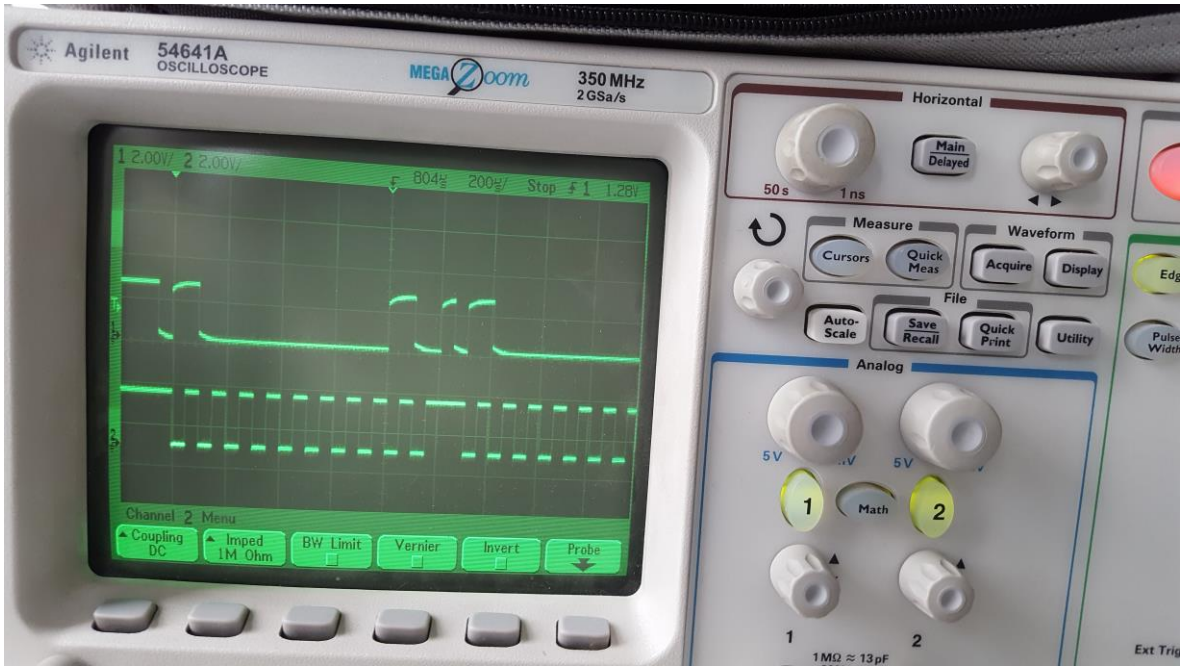
[4] https://axotron.se/blog/crosstalk-problems-when-running-i2c-signals-in-a-cable/

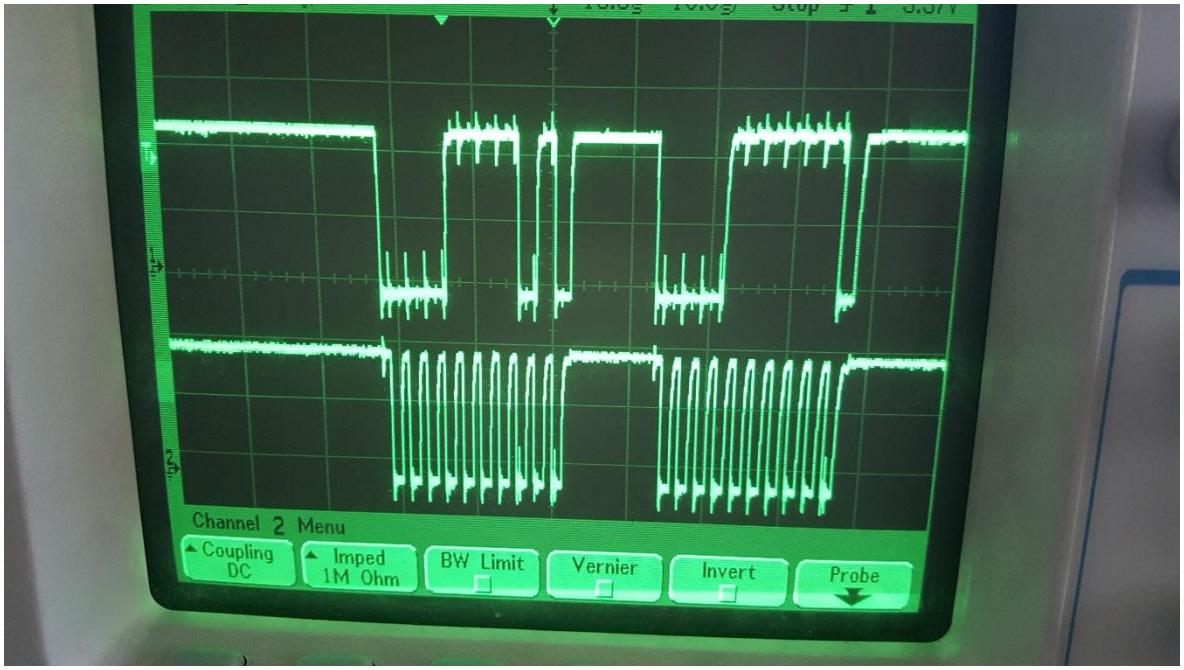

V(scl)
V(sda)
3.3V
3.0V
2.7V
2.4V
2.1V
1.8V
1.5V
1.2V
0.9V
0.6V
0.3V
0.0V
0.0µs
0.8µs
1.6µs
2.4µs
3.2µs
4.0µs
4.8µs
5.6µs
6.4µs
7.2µs
8.0µs

6.7 I2C-Bus Timingdiagramme

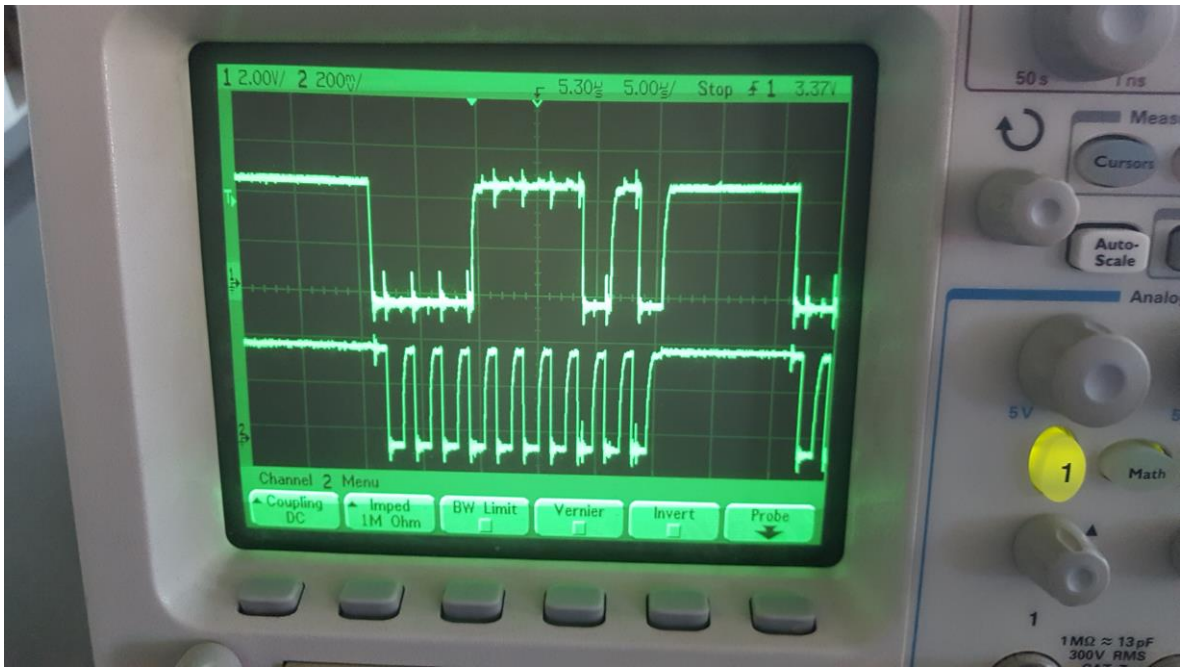


Vergrößerung:





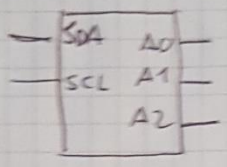
Vergrößerung:



6.8 Beispiel TMP275 Temperatursensor

Bsp. I²C-Bus Sensor

TMP 275



Slave Adresse | 1 | 0 | 0 | 1 | A2 | A1 | A0 |

1) Konfigurationsregister beschreiben

2 Byte schreiben

						P1	P0
0	0	0	0	0	0	0	1

OS	R1	R0	F1	F0	POL	TM	
0	0	0	0	0	0	0	1

9 Bit Auflösung 0,5° C

not used

Standby

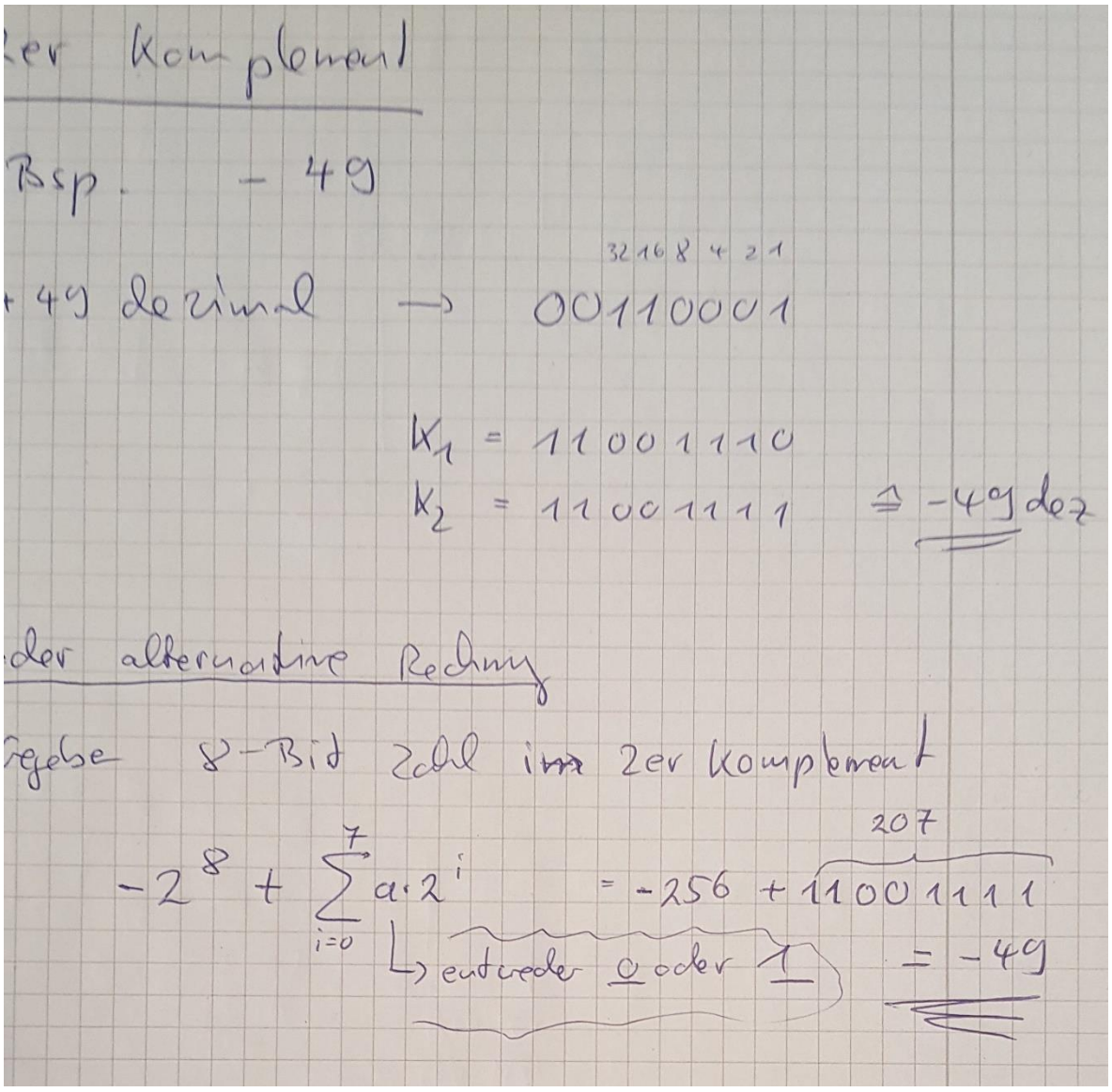
2) Temperatur lesen

T11									T3	T2	T1	T0			

Bsp.:

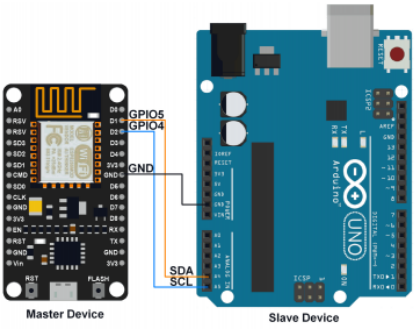
0	0	0	0	0	0	0	0	0				
+49	0	0	1	1	0	0	0	1				
122	0	1	1	1	1	0	1	0				
+122,5	0	1	1	1	1	0	1	0	1	0	0	0
-49	1	1	0	0	1	1	1	1	0	0	0	0

6.8.1 2er Komplement



6.9 Beispiel I2C-Bus NodeMCU und Arduino

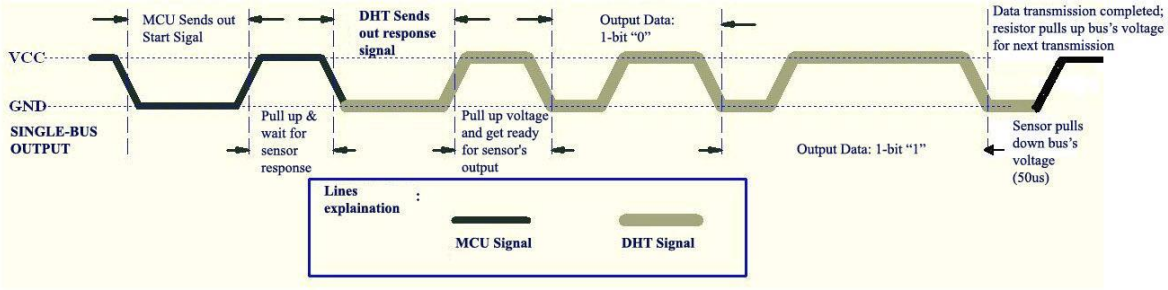
Das nebenstehende Bild zeigt I2C-Bus Verbindung zwischen einem NodeMCU und einem Arduino Board[5] (z.B. Nucleo STM32 wie im Praktikum)



[5] https://www.ob121.com/doku.php?id=de:esp:nodemcu_i2c

7 Proprietäre Schnittstelle (Beispiel DHT11 Sensor)

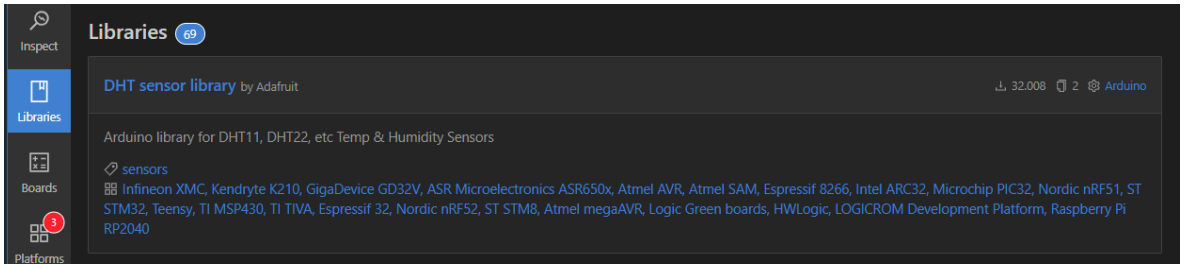
Es gibt viele proprietäre serielle Schnittstellen, ein Beispiel dafür ist der Temperatur-Feuchtesensor DHT11 mit einer sehr einfachen 1 Drahtschnittstelle.



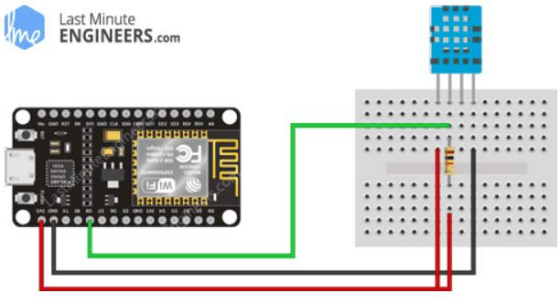
Auszug aus dem Datenblatt:

Data consists of decimal and integral parts. A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.
Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

Die Daten sind programmtechnisch sehr einfach aufzubereiten, aber am einfachsten natürlich mit bereits existierenden Bibliotheken wie z.B. für Arduino:



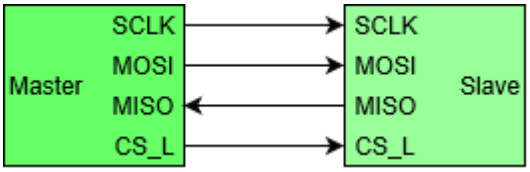
Die Verkabelung mit einem NodeMCU ist einfach (ca. 5 – 10 kOhm für den Pull-Up Widerstand):



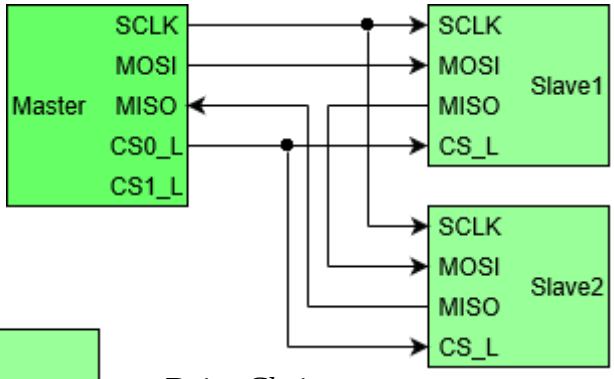
Wiring DHT11 Temperature Humidity Sensor with ESP8266 NodeMCU

8 SPI-Bus

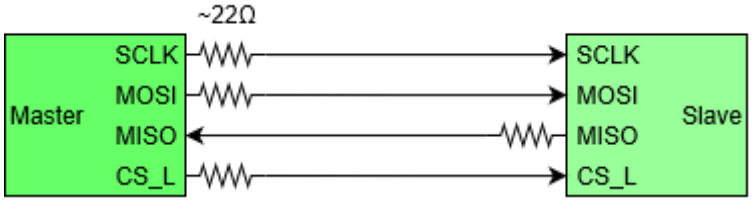
- **SPI** oder **Serial Peripheral Interface**
- Motorola
- Master Slave
- serieller Datenbus
- Vollduplex
- Point 2 Point
- kurze Distanzen ca. bis 50 MHz
- einfach zu programmieren
- weit verbreitet für serielle Datenübertragung zw. uC und
  - EEPROM
  - RTC
  - Sensoren
  - Displays



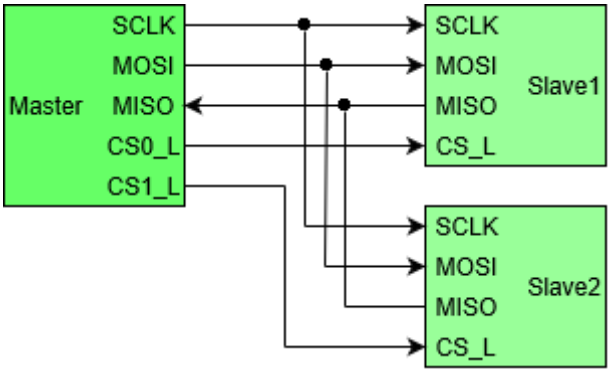
Master Slave



Daisy Chain



High Speed Design[6]



8.1 Prinzip Schieberegister

[6] https://practicalee.com/spi/

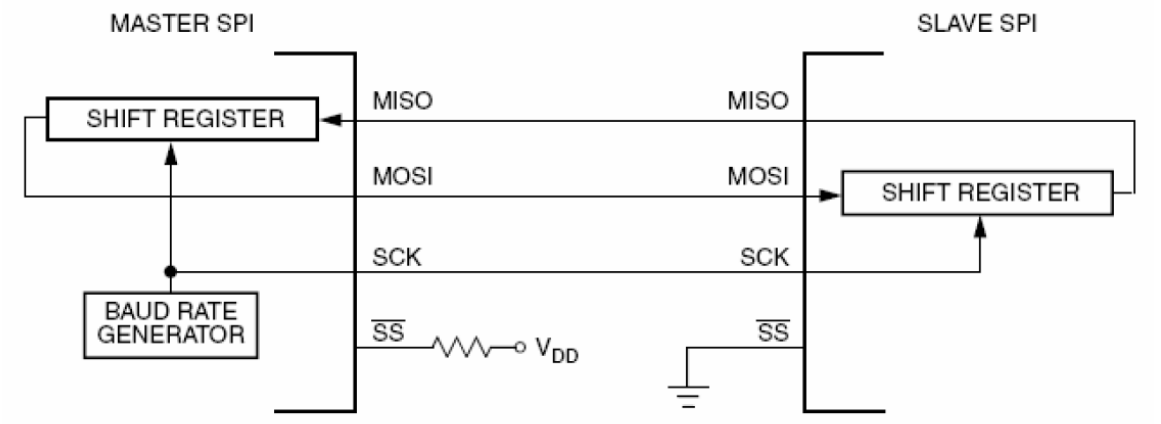
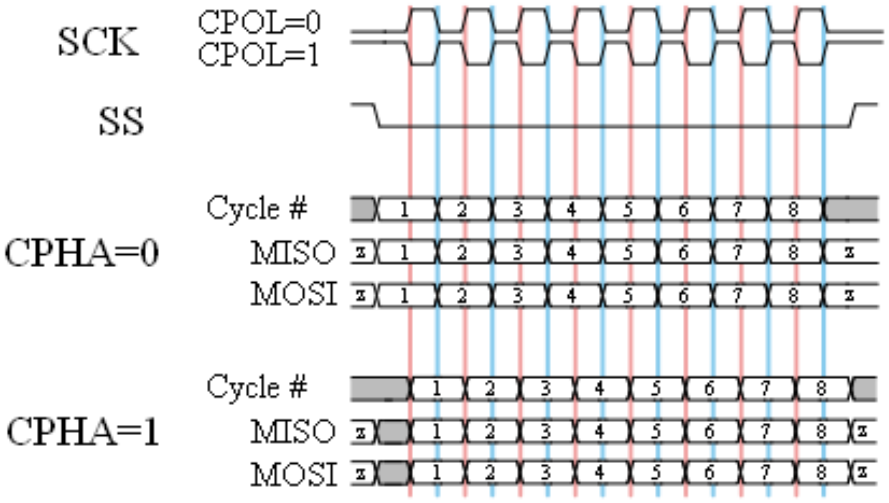


Bild [4]: Datenübertragung zwischen Master und Slave
Quelle: Freescale SPI Handbuch

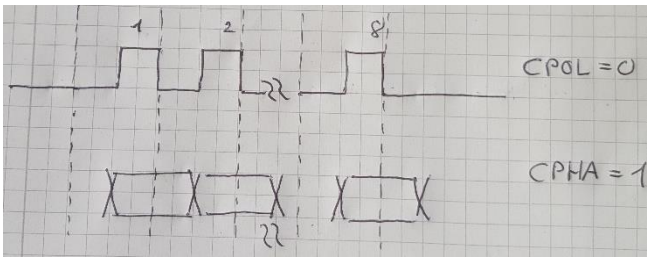
8.2 Zuordnung Takt zu Datenleitung

Die Zuordnung zwischen einem Takt und der stabilen Datenleitung erfolgt über CPOL und CPHA.



Das obige Bild wird deutlicher an einem konkreten Beispiel für CPOL=0 und CPHA=1:

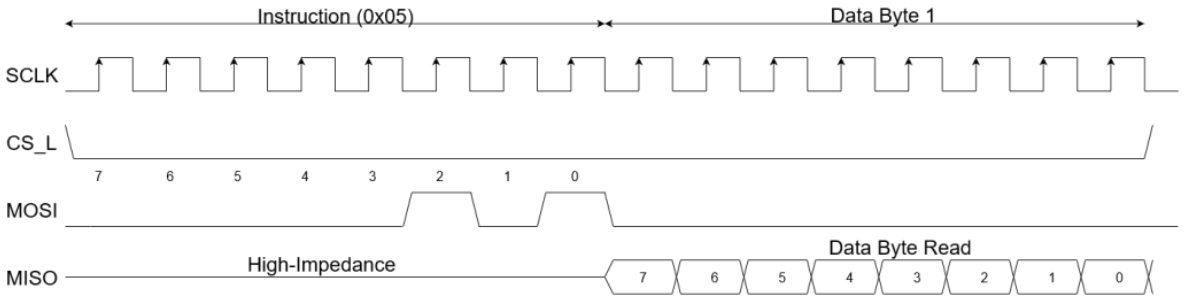
CPOL =0 bedeutet dass im Ruhezustand die Taktleitung auf Lowpegel ist. CPHA =1 bedeutet dass die Datenleitungen MOSI/MISO bei der 2ten Taktflanke stabil sind.



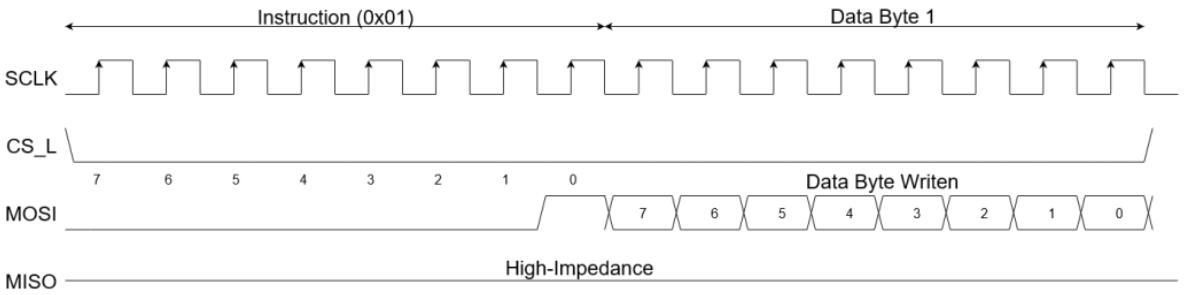
8.3 Beispiel für Protokoll

In dem Beispiel ist der SPI Modus 0 eingestellt (CPOL=0, CPHA=0)

8.3.1 Lesen

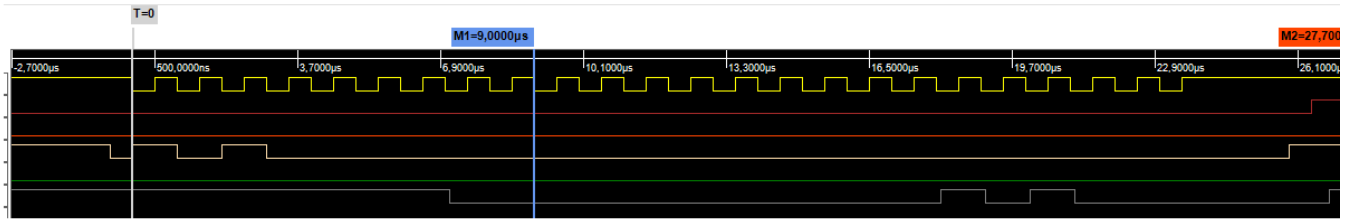


8.3.2 Schreiben



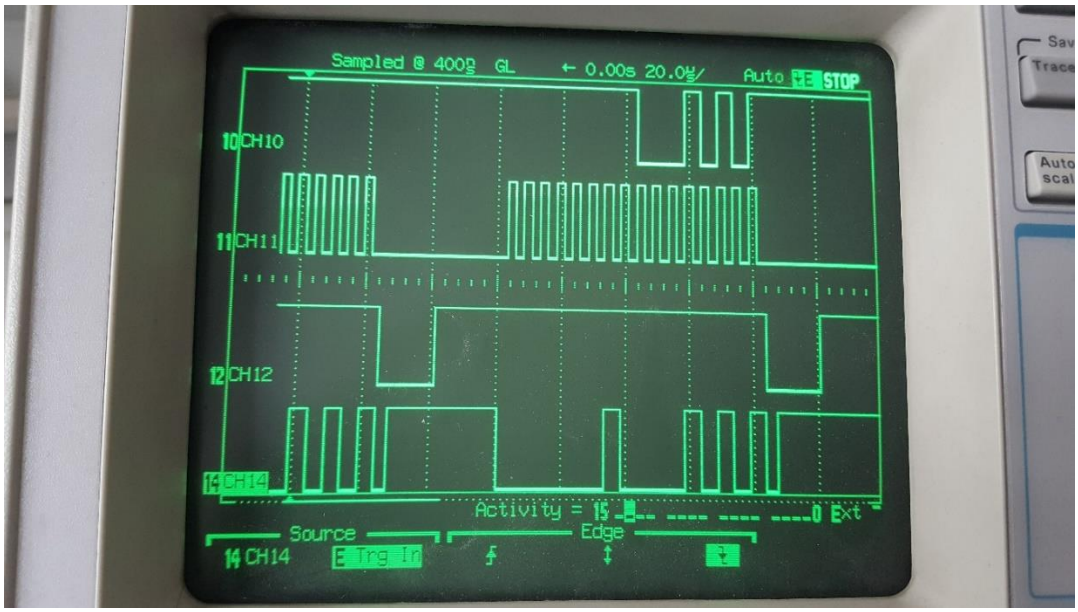
8.3.2.1 Beispiel 2

Das folgende Bild zeigt einen Ausschnitt aus einer typischen SPI Bus Datenkommunikation



- Mit welcher Taktfrequenz wird übertragen?
- Wie viele Bytes werden gesendet bzw. empfangen?
- Geben Sie die Werte der gesendeten und empfangenen Byts an.
- Mit welchem Modus wird übertragen (CPOL und CPHA)?
- Bezeichnen Sie die jeweiligen Signale.
-

8.3.2.2 Beispiel 3:



8.3.2.3 Beispiel 4

Trinamic TMC5160 Schrittmotor-Controller

SPI DATAGRAM STRUCTURE				
MSB (transmitted first)	40 bit			LSB (transmitted last)
39 …				… 0
→ 8 bit address **← 8 bit SPI status**	**← → 32 bit data**			
39 … 32	31 … 0			
→ to TMC5160 RW + 7 bit address ← from TMC5160 8 bit SPI status	8 bit data	8 bit data	8 bit data	8 bit data
39 / 38 … 32	31 … 24	23 … 16	15 … 8	7 … 0

w	38…32								31…28				27…24				23…20				19…16				15…12				11…8				7…4				3…0			
39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

9 Vergleich I2C-/SPI Bus

- SPI
  - Geeignet für Kommunikation auf Boardebene mit niedrigen Distanzen
  - Vollduplex – beide Parteien können gleichzeitig senden
  - Hohe Geschwindigkeit bis typischerweise 10 MHz
  - Slave Select wird für die Auswahl von Komponenten verwendet
  - Drei Leitungen und mehrere Slave-Select Leitungen bei Sterntopologie notwendig (viele Pins am Microcontroller benötigt!)
- I2C
  - Geeignet für Kommunikation auf Boardebene mit niedrigen Distanzen
  - Halbduplex – nur eine Partei kann gleichzeitig senden
  - Geringe Geschwindigkeit bei typischerweise 400 KHz
  - Jeder Baustein benötigt eindeutige Adresse
  - Nur zwei Leitungen notwendig

Sonstige Vorteile des SPI-Busses:
- Einfach per Software im Master zu implementieren
- gut geeignet für Kommunikation in kleinen Systemen

Sonstige Nachteile des SPI-Busses:
- Keine Kontrolle der Flankensteilheit der Signale
- Keine Fehlerabsicherung bei der Übertragung

Sonstige Vorteile des I²C-Busses:
- der Master generiert den Bustakt
- Jedes Gerät am Bus hat seine einzigartige Adresse

Sonstige Nachteile des I²C-Busses:
- Sehr anfällig für Störgeräusche

11 CAN-Bus

Der CAN-Bus wurde in den 80er Jahren von der Firma Bosch entwickelt und hat sich nicht nur in der Automobilindustrie sondern auch in der Automatisierungstechnik sehr stark verbreitet. Anbei einige wichtige Eigenschaften:

- Ereignisgesteuert: Jeder Teilnehmer entscheidet, wann er ein Telegramm sendet
- CSMA/CR: Zugriffsverfahren ->**C**arrier **S**ense **M**ultiple **A**ccess /**C**ollision **R**esolution (/CD Collision Detection z.B. bei Ethernet LAN und /CA Collision Avoidance bei WLAN)
- Keine klassische Adressierung, durch einen sogenannten **Identifier** ist jede Nachricht gekennzeichnet und in der Regel Inhalt und Absender zugeordnet
- Prioritätsgesteuerte Nachrichten: Der Identifier legt die Priorität fest
- Synchronisationsmöglichkeit: Übergang von rezessiv zu dominantem Spannungspegel
- Fehlererkennung: 15 Bit CRC, Bit Stuffing

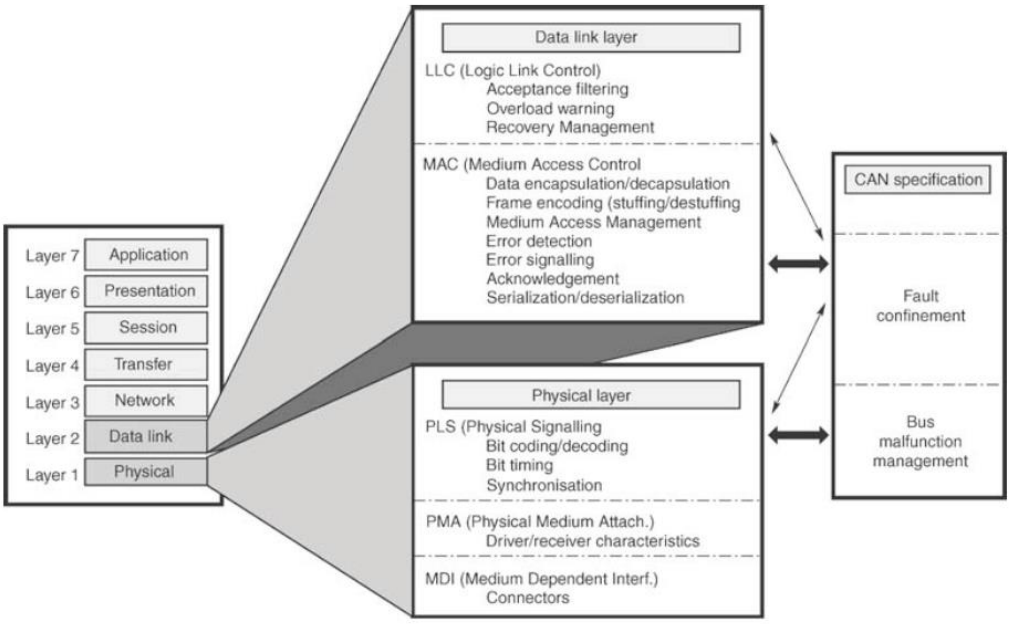
Weitere Eigenschaften:

- Broadcastprinzip: Nachrichten gehen an alle Teilnehmer
- Automatische Wiederholung gestörter Nachrichten (abhängig vom CAN-Controller)
- Fehlertoleranz: Fehler werden klassifiziert in vorübergehend und dauernd, Abschalten des jeweiligen Knotens (Error-Frame)

Einsatzgebiet neben Automobil/LKW/Bus

- Stellglieder, Motoren etc.
- Luftfahrttechnik, siehe ARINC825

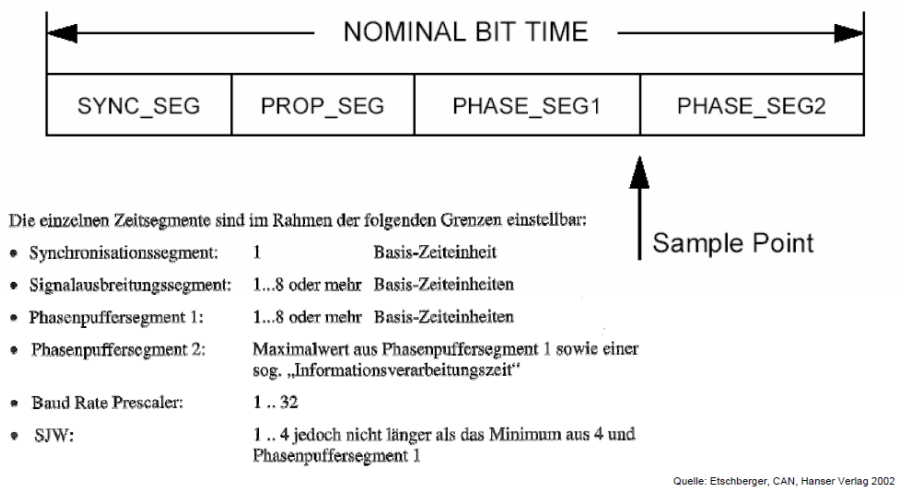
11.1 CAN-Bus im Schichtenmodell



Der CAN-Bus kann also in die unteren beiden Schichten des OSI Schichtenmodells eingeordnet werden.
In der Applikationsschicht findet sich dann das Protokoll CAN-Open, in der Automatisierungstechnik ist dieses Protokoll z.B. zur Ansteuerung von Motoren, Stellgliedern etc. weit verbreitet.

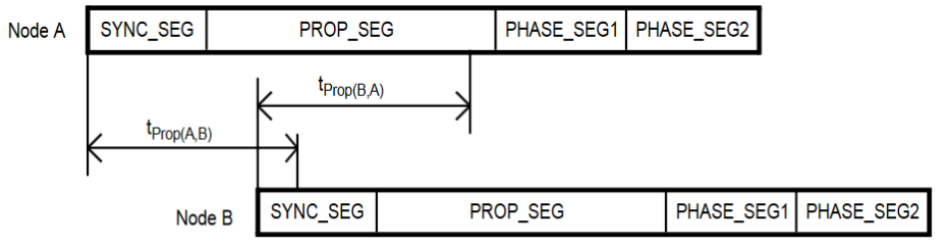
11.2 Bitabtastung

Die Dauer eines Bits hängt von den Timingeinstellungen des Controllers ab. Diese unterteilen sich in mehrere Segmente, welche in der folgenden Abbildung dargestellt sind. Das Signal wird zum Zeitpunkt des Sampling Point auf dem Bus abgetastet. Dieser Punkt muss sich möglichst weit am Ende der nominalen Bitzeit befinden, um genügend Zeit für die Ausbreitung zu erlauben.

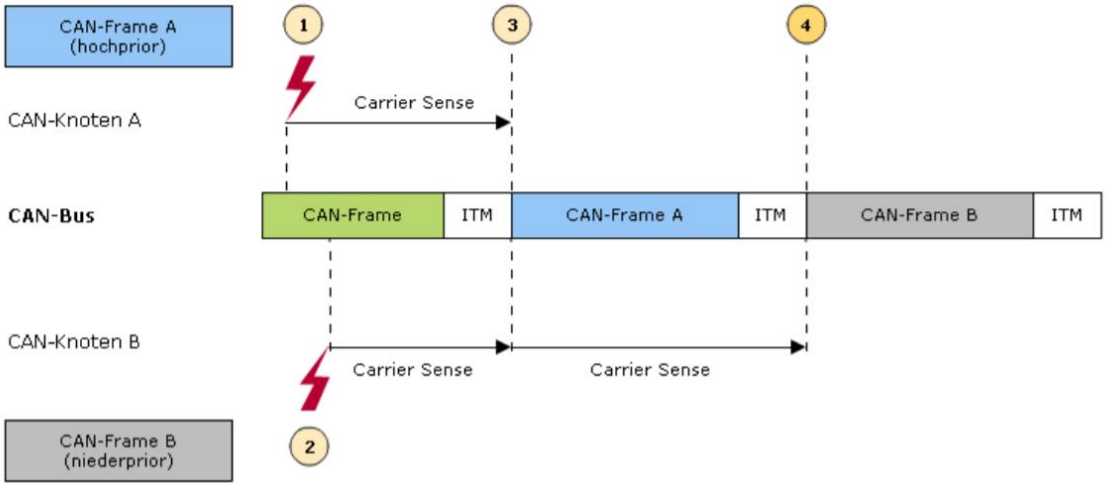


Die Synchronisation-Jump-Width (SJW) gibt an, um wie viele Basis-Zeiteinheiten (Time-Qanta) der Sampling Point nach vorne oder hinten verschoben wird, falls eine Neusynchronisierung an einer Flanke nötig ist. Dabei wird das Phase Buffer Segment 1 und 2 entsprechend verkürzt bzw. verlängert. Hierdurch können kleinere Differenzen der Oszillatoren von CAN Controllern ausgeglichen werden. Während des Propagation Segment breitet sich das Signal entsprechend auf dem Bus und durch alle Transceiver sowie Controller und zurück aus.

Wenn zwei Teilnehmer gleichzeitig senden müssen Sie dies innerhalb einer Bitzeit erkennen, das offensichtliche Problem dabei ist die endliche Laufzeit des Signals auf der Leitung. Das folgende Diagramm zeigt dies, das Signal des Knotens B muss vor dem Sampling Point wieder beim Knoten A ankommen:

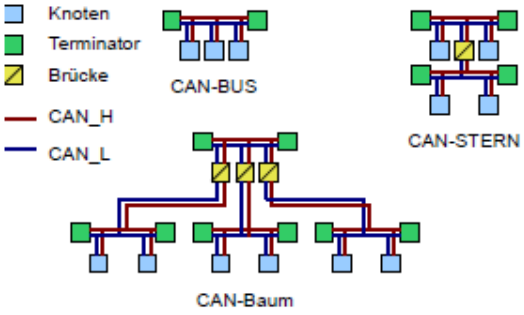


Das folgende Bild zeigt den Ablauf wenn zwei Teilnehmer Daten auf den Bus senden wollen. Der Knoten B kann erst nachdem Knoten A den Frame gesendet hat erfolgreich auf den Bus zugreifen.



11.3 Topologie

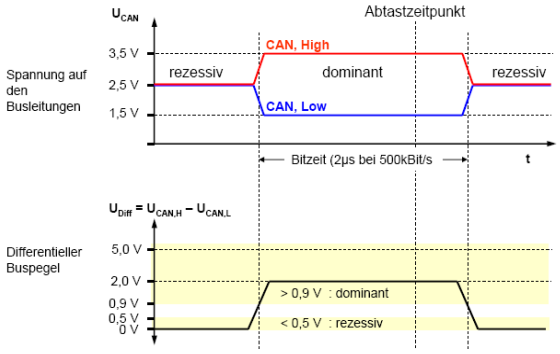
Typischerweise wird eine Bustopologie mit Abschlusswiderständen an beiden Enden der differentiellen Leitung angewendet. Der Abschlusswiderstand auf beiden Seiten der Leitung beträgt 120 Ohm.



Über Mikrocontroller mit zwei CAN-Controllern sind selbstverständlich auch Stern und Baumtopologie möglich.

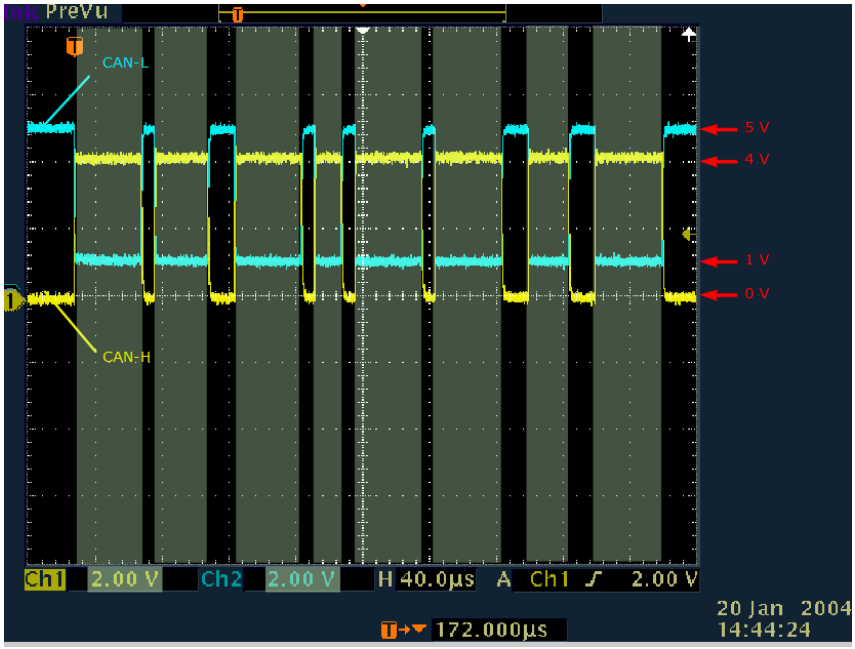
11.4 Buspegel

Es handelt sich um eine differentielle Übertragung.
Das unten gezeigte Oszilloskopbild zeigt eine typische CAN-Bus Übertragung.



Typische Fragen:

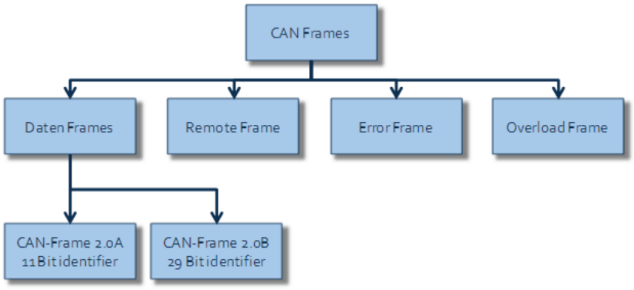
- Womit beginnt hier die Übertragung, dominant oder rezessiv.
- Welche Datenrate?



11.5 CAN-Bus Frames

11.5.1 Welche Frames gibt es?

Der Benutzer ist für die Daten- und Remoteframes verantwortlich. Die Error-und Overloadframes werden vom CAN-Controller ausgelöst.
In der Praxis werden Remote-Frames äußerst selten verwendet.



11.5.1.1 Daten Frame

Das Bild zeigt einen mit einem CAN-Analyser aufgezeichneten Daten Frame:

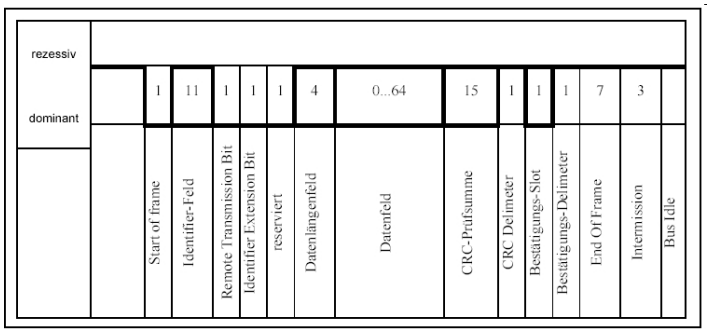
Receive Overruns: 0 Errors: 0

No	Time (abs)	State	ID (he...	DLC	Data (hex)	ASCII
102	00:08:26.130		321	6	48 61 6C 6C 6F 00	Hallo.
103	00:08:31.149		321	6	48 61 6C 6C 6F 00	Hallo.

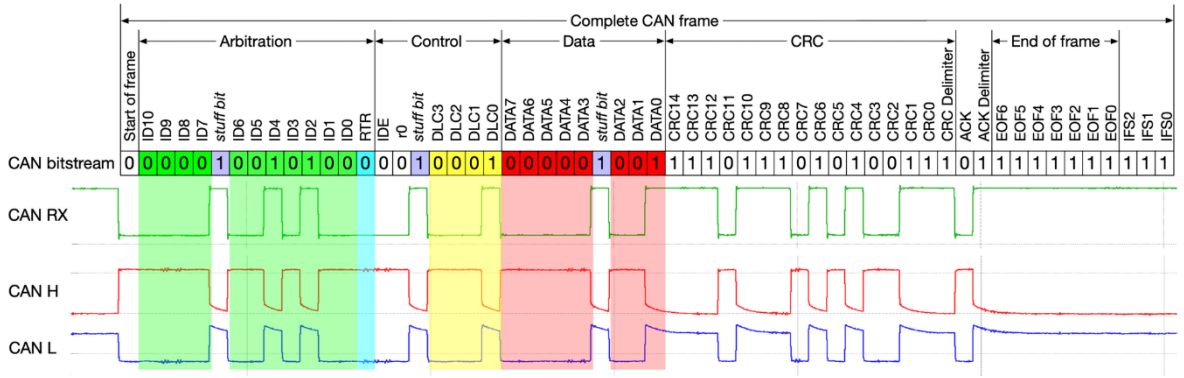
Eine CAN-Bus Nachricht besteht aus mehreren Komponenten:

- State: Standard oder Extended Identifier (E = Extended)
- ID im Hexadezimalformat
- DLC: wie viele Bytes werden übertragen
- Datenfeld: Daten die übertragen werden

11.5.2 Standard Frame (2.0A)

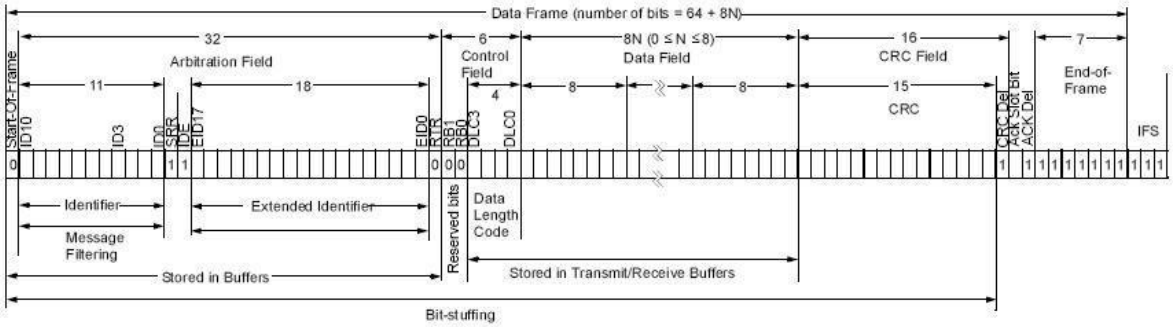


Das folgende Bild aus Wikipedia zeigt den Standard-Frame mit eingezeichneten Spannungspegeln:



Interessant sind bei diesem Bild vor allem die Stuffbits sowie die Erzeugung von CAN_RX aus CAN_H und CAN_L. Eine einfache Prüfungsfrage wäre die Bestimmung des Identifiers und des Datenworts.

11.5.3 Extended Frame (2.0B)



Welches Bit identifiziert den Standard- bzw. den Extended-Frame?

11.6 Fehlerarten

Der CAN Standard beschreibt 5 Fehler, die bei der Übertragung auftreten und detektiert werden können:

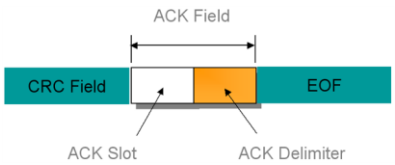
- **Bit Error:** Jeder Controller überwacht während des Sendevorgangs den Bus und kann so erkennen, wenn ein Bit nicht so übertragen wird, wie es vorgesehen ist. Dies ist der Fall wenn beispielsweise ein rezessives Bit gesendet wird, jedoch ein dominantes zur selben Zeit am Bus anliegt. Eine Ausnahme hierbei bildet das Arbitrierungsfeld sowie der ACK Slot.
- **Stuff Error:** Tritt auf wenn 6 mal in Folge ein gleichwertiges Bit übertragen wurde. Dies wird im Normalfall durch Bit stuffing verhindert.
- **CRC Error:** Sollte die im Empfänger berechnete Prüfsumme nicht mit der gesendeten übereinstimmen, tritt dieser Fehler auf.
- **Form Error:** Tritt auf falls der Aufbau eines CAN Frames nicht dem Standard entspricht.
- **Acknowledgment Error:** Der Sender erwartet im ACK Slot ein dominantes Bit, welches von mindestens einem anderen Teilnehmer stammt, der den Frame erfolgreich empfangen hat. Tritt dies auf hat voraussichtlich kein weiterer Teilnehmer die Nachricht empfangen.

Was geschieht wenn einer der Fehler auftritt?

- Jeder der angeschlossenen CAN-Controller sendet einen Error Frame.
- Je nach der Konfiguration des CAN-Controllers wird die Nachricht schnellstmöglich wieder gesendet.

11.6.1 Acknowledge

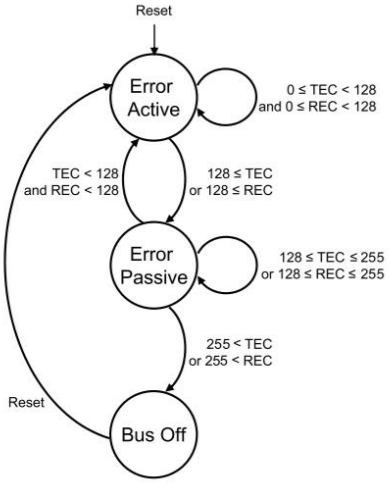
Das ACK Feld besteht aus 2 Bits, dem ACK Slot und dem ACK Delimiter. Der Producer des Frames sendet beide Bits des ACK Feldes recessiv. Jeder!!!! Consumer, der eine korrekte Nachricht empfangen hat, teilt dies durch eine dominantes Bit innerhalb des ACK Slots mit.



Wenn kein Teilnehmer außer dem senden Knoten am Bus ist werden Errorframes gesendet

11.6.2 Error Frames

Der CAN-Controller kennt drei unterschiedliche Zustände:



Error Active: Dies ist der normale Zustand, der CAN-Controller kann Nachrichten empfangen und senden. Tritt ein Fehler auf wird ein sogenanntes „Active Error Flag“ gesendet, bestehend aus 6 dominanten Bits.

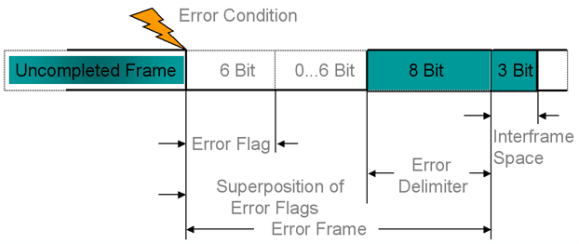
Error Passive: Dieser Zustand wird erreicht, wenn er bereits mehrere Fehler auf dem Bus lokalisiert hat. Er sendet dann 6 rezessive Bits.

Bus Off: Bei diesem Zustand wird der Netzteilnehmer komplett vom Bus getrennt und nimmt nicht mehr an der Kommunikation teil. Dieser Zustand kann nur durch einen Reset des CAN-Controllers behoben werden.

TEC = Transmit Error Counter
REC = Receive Error Counter

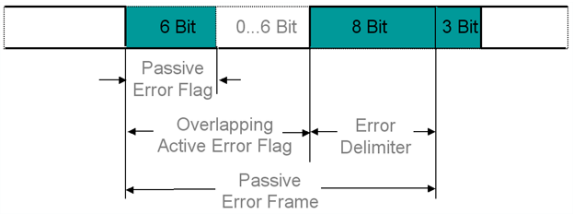
11.6.2.1 Active Error Frame

Das Error Flag besteht aus 6 dominanten Bits. Durch die Verletzung der Bitstuffing Regel wird damit der Fehler öffentlich gemacht.-



11.6.2.2 Passive Error Frame

Das Error Flag besteht aus 6 rezessiven Bits



Beispiel für die Fehlersignalisierung bei STM32 CAN-Controller. In dem gezeigten Ausschgitt des CAN Error Statusregisters können die Fehler gezielt beobachtet und ausgewertet werden:

Bits 31:24 **REC[7:0]:** Receive error counter
The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

Bits 23:16 **TEC[7:0]:** Least significant byte of the 9-bit transmit error counter
The implementing part of the fault confinement mechanism of the CAN protocol.

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **LEC[2:0]:** Last error code
This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to '0'.
The LEC[2:0] bits can be set to value 0b111 by software. They are updated by hardware to indicate the current communication status.
000: No Error
001: Stuff Error
010: Form Error
011: Acknowledgment Error
100: Bit recessive Error
101: Bit dominant Error
110: CRC Error
111: Set by software

Bit 3 Reserved, must be kept at reset value.

Bit 2 **BOFF:** Bus-off flag
This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255, refer to *Section 29.7.6 on page 829*.

Bit 1 **EPVF**: Error passive flag
This bit is set by hardware when the Error Passive limit has been reached (Receive Error Counter or Transmit Error Counter>127).

Bit 0 **EWGF:** Error warning flag
This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter≥96).

11.7 Stuffbits

Problem:

- Stille auf dem Medium muss erkennbar sein (eine lange Folge von 1-Bits (elektrisch null) kann als Stille fehlinterpretiert werden)
- Lange Folgen gleicher Bits erschweren Synchronisation der Stationen, da es keine Bitflanken gibt
- Mehr als 5 mal 0 in Folge wird als Errorframe benutzt - Unterscheidung ist notwendig

Die Lösung ist das „Bitstuffing“: Nach 5 gleichen Bits fügt das Protokoll ein Bit des anderen Wertes ein, der Empfänger entfernt dann nach 5 identischen Bits das folgende Bit.

11.7.1 Anzahl Stuffbits

Trotz des selbsttätigen Zugriffs eines CAN-Knotens auf die Busleitung können für einen Knoten der höchsten Priorität Anhaltswerte für die effektive Übertragungsrate angegeben werden.
Eine Nachricht im Standard-Format mit acht Datenbytes benötigt maximal 130 Bits. Dabei geht man von einer maximalen Anzahl von 19 Stuff-Bits und 3 Zwischenraumbits aus:

1 Start bit
+11 Identifier bits
+ 1 RTR bit
+ 6 Control bits
+ 64 Data bits
+ 15 CRC bits
+ 19 (maximum) Stuff bits
+ 1 CRC delimiter
+ 1 ACK slot
+ 1 ACK delimiter
+ 7 EOF bits
+ 3 IFS (Inter Frame Space) bits = 130 bits

Im Extended-Format sind maximal 154 Bits zu übertragen.

Die maximale Übertragungsdauer C_i einer Nachricht lässt sich folgendermaßen bestimmen:

$$C_i = \Bigg(\text{Header/CRC-Länge} + 8 \cdot \text{Anzahl Daten-Bytes} + 13 + \underbrace{\left\lfloor \frac{\text{Header/CRC-Länge} + 8 \cdot \text{Anzahl Daten-Bytes} - 1}{4} \right\rfloor}_{\text{Anzahl Stuff-Bits}}\Bigg)\tau_{\text{bit}} \qquad (8.6)$$

Die Header/CRC-Länge hängt in dieser Gleichung von der Art der CAN-Nachricht ab. Bei CAN-Identifiern mit 11 Bit beträgt die Länge 34 Bit und bei 29 Bit Identifiern beträgt die Header/CRC-Länge 54 Bit. Die 13 Bits in der Gleichung ergeben sich aus den folgenden Bits:

- 1 CRC Delimiter
- 1 ACK Slot
- 1 ACK Delimiter
- 7 EOF Bits
- 3 IFS (Inter Frame Space) Bits

11.8 Buslast Berechnung

Da es sich um einen nicht deterministischen Bus handelt ist dies nur eine ungefähre Berechnung.

Nachfolgend ein Beispiel unter Verwendung eines Standard-Identifiers:

- 1 bit start bit
- 11 bit identifier
- 1 bit RTR
- 6 bit control field
- 0 to 64 bit data field
- 15 bit CRC
- Bit stuffing is possible in the above, for every sequence of 5 consecutive bits of same level. Somewhere around 18 bits in the worst case.
- 3 bit delimiter, ack etc.
- 7 bit end of frame
- 3 bit intermission field after frame

Der CAN Bus Frame enthält also ungefähr 125 Bit.

Angenommen ist eine Bitrate von 500 kBit/s:

Bit-Zeit = 1 / Bit-Rate = 1 / (500 * 1000) s = $2 * 10^{-6}$ s = 2 µs

1 Bit benötigt also 2 µs für die Übertragung bei 500 kBit/s.

1 Frame benötigt (2 µs/bit * 125 bit) = 250 µs.

Die Buslast für 1 Nachricht alle 100ms mit 500 kBit/s kann so berechnet werden:

In 100 ms wird der Bus für 250 µs belegt. Die Buslast ist:

250 µs / 100 ms = (250 / (100*1000)) * 100 % = 25000 / 100000 % = 0.25 %

Dies kann auch für unterschiedliche Intervalle berechnet werden, z.B.:

1 Frame alle 10 ms = 100 Frames alle 1000 ms
1 Frame alle 100 ms = 10 Frames alle 1000 ms
1 Frame alle 1000 ms = 1 Frame alle 1000 ms

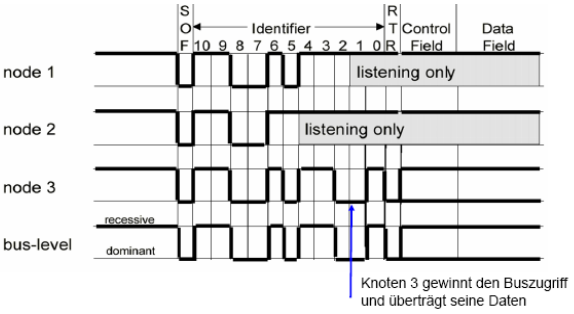
Gesamt: 111 Frames alle 1000 ms

Gesamtzeit auf dem Bus 111 * 250 µs

Gesamtzeit: 1000 ms = 1000 * 1000 µs

Buslast: ((111 * 250) / (1000 * 1000)) * 100 % = 2.775 %

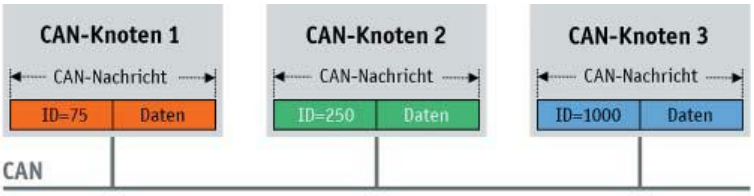
11.9 Arbitrierung

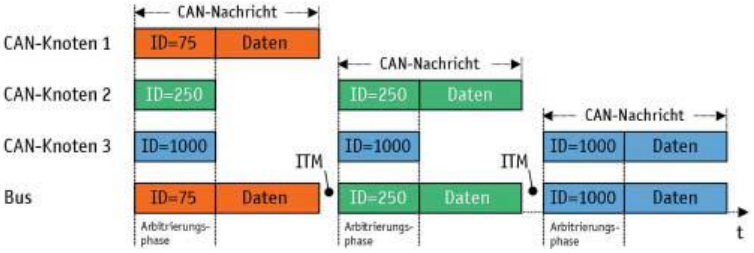


- Alle CAN-Bus-Tranceiver (Treiber) eines Netzwerkes sind als hartverdrahtete ODER-Verschaltung ausgelegt.
- Wenn mehrere Tranceiver (Busknoten) gleichzeitig versuchen einen dominanten und/oder rezessiven Buspegel einzustellen, überschreibt bei gleichzeitiger Sendung ein dominantes Bit immer ein rezessives. Die Logik im Schnittstellenbaustein der rezessiven Teilnehmer erkennt dies als Arbitrationsverlust und zieht seinen Sender für die Dauer des anliegenden Telegramms (Botschaft) vom Bus zurück.
- Die 11 Bit Identifiers des Bildes sind
  - 110 0101 1111 = 0x65F
  - 110 0111 1111 = 0x67F
  - 110 0101 1001 = 0x659

Es gewinnt immer der Knoten mit dem geringsten Wert des Identifiers.
Wenn die Werte als „normale“ Dulzahlen interpretiert werden, dann gilt „MSB first“.

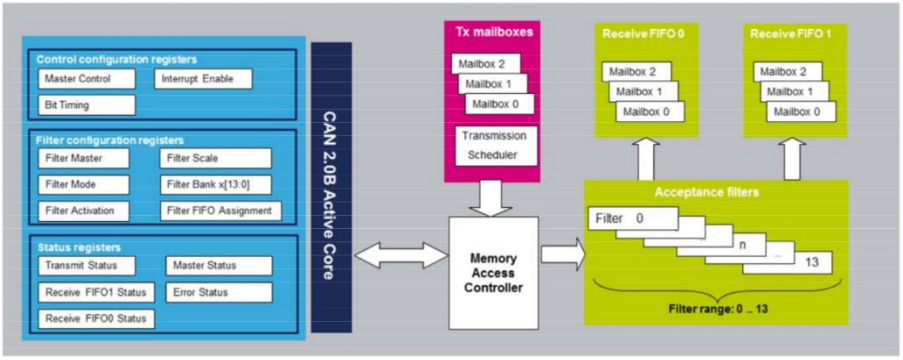
Das folgende Bild zeigt den Ablauf bei der Arbitrierung an einem weiteren Beispiel:





11.10 Senden/Empfangen

Das Blockschaltbild für Senden und Empfangen zeigt das nebenstehende Bild. Für das Senden sind 3 Mailboxen verfügbar. Das Empfangen der Daten erfolgt in zwei FIFOs die jeweils 3 Mailboxen enthalten.



11.10.1Senden

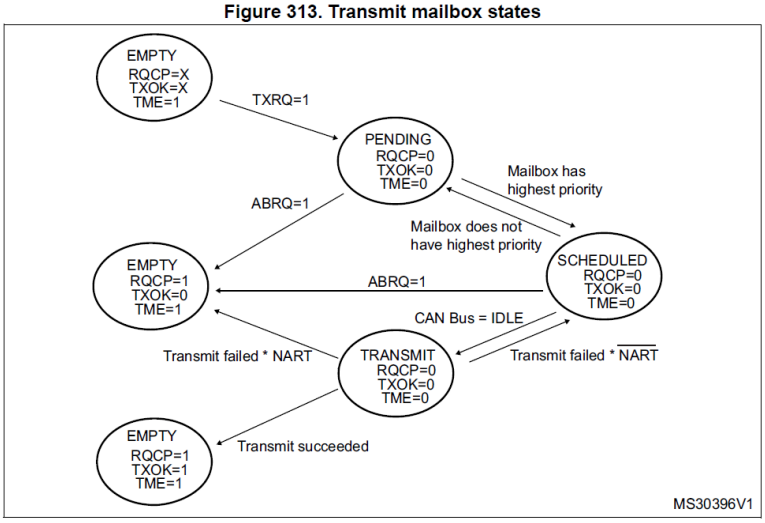
- **Wer kann senden?:** Jeder Knoten kann senden, der Bus muss lediglich für die geforderte Dauer im Idle- Zustand sein.
    1. Alle weiteren Knoten beginnen zu empfangen mit Ausnahme des sendenen Knotens (das senden und empfangen geschieht gleichzeitig)
    2. Wenn nun ein weiterer Knoten mit der Übertragung beginnen will startet der Arbitrierungsprozess….lediglich der Knoten mit der höchstprioren ID kann seine Nachricht übertragen.
- **ACK:** Wenn der sendende Knoten den Sendeprozess beendet hat wartet er eine Bitzeit für die Bestätigung im ACK Feld ab. Falls die Nachricht fehlerfrei übermittelt wurde, wird dieses Feld von anderen Knoten (üblicherweise von allen angeschlossenen Knoten) auf 0 gezogen.
- Falls ACK bestätigt wird geht der Sendeknoten davon aus, dass die Nachricht die Empfänger erreicht hat und geht in den Empfangsmodus oder beginnt mit dem senden einer weiteren Nachricht. Zu diesem Zeitpunkt kann jeder beliebige Knoten mit dem Senden von Nachrichten beginnen, andernfalls geht der Bus in den Idle Zustand. Alles beginnt wieder von vorne.
- **NOT ACK:** Wenn ACK nicht bestätigt wird versucht der Sendeknoten die Nachricht zum frühestmöglichen Zeitpunkt erneut zu übertragen. Dies wird eine Endlosschleife,

wenn ACK niemals bestätigt wird (dies ist aber abhängig vom CAN-Controller bzw. der Konfiguration desselben.

Was geschieht, wenn kein Knoten die gesendete Nachricht verwendet? Ganz einfach, es passiert nichts, der sendende Knoten überprüft lediglich ob das ACK bestätigt wird.

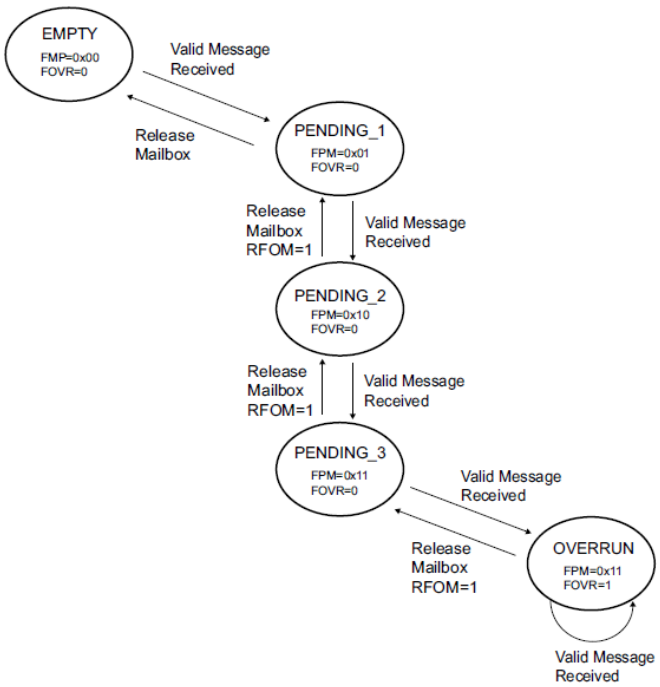
Nebenstehendes Bild zeigt das Ablaufdiagramm am Beispiel des STM CAN-Controllers mit drei Sende Mailboxen

Figure 313. Transmit mailbox states



11.10.3Empfangen und Akzeptanzfilterung

1. Alle Knoten ausgenommen des sendenden Knotens sind im Listen Modus.
2. Der gesendete Frame wird von allen Knoten empfangen. Falls die Nachricht fehlerfrei ist wird das ACK Bit gesetzt (dominant = 0).
3. Der Frame läuft beim Empfänger durch das Akzeptanzfilter und wird je nach Einstellung des Filters geblockt oder durchgelassen. Falls durchgelassen landet der Frame im FIFO des CAN-Controllers. Das Bild zeigt den Ablauf am Beispiel eines STM32 Mikrocontrollers. Dieser CAN Controller hat zwei Empfangs FIFOs mit jeweils Platz für drei Nachrichten:



4. Der Mikrocontroller wird informiert, dass Daten im FIFO sind und muss die Daten schnellstmöglich lesen. Dabei ist im einfachsten Fall Polling möglich, dabei ist aber der Mikkrocontroller blockiert, deshalb wird bevorzugt das interruptgesteuerte Lesen verwendet.
5. Was dann mit der Nachricht geschieht ist in der Verantwortung der Benutzersoftware und unabhängig vom CAN-Controller.

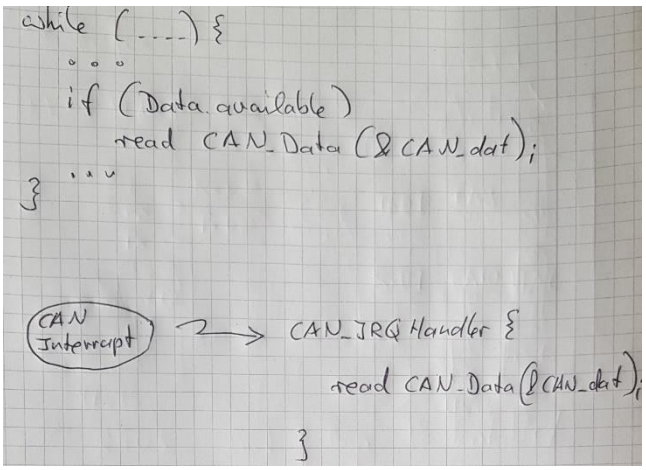
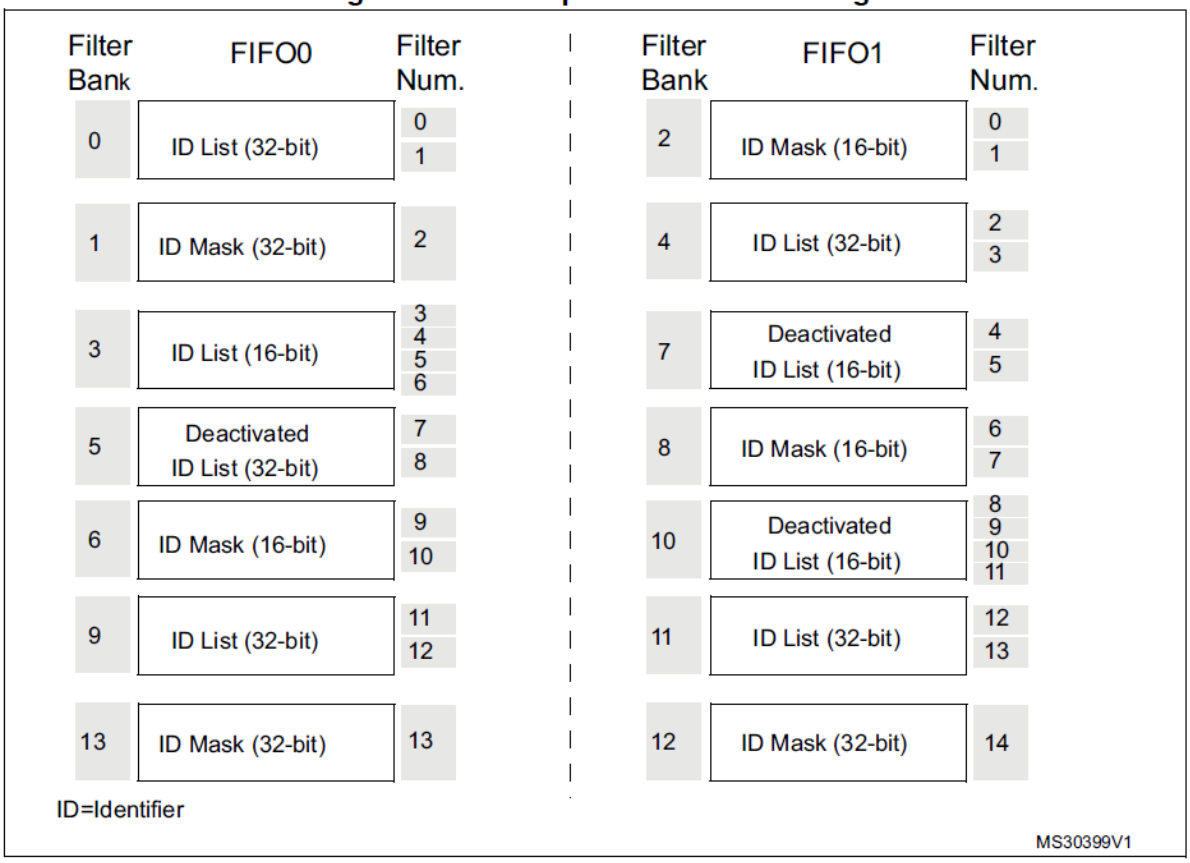


Figure 316. Example of filter numbering



Mask Mode:

Die **Filter Maske** spezifiziert diejenigen Bits, die mit der **Filter ID** verglichen werden:
Bei der Filtermaske lässt eine 0 das entsprechende Identifierbit passieren.

- Wenn ein Maskenbit auf Null gesetzt ist, wird das entsprechende ID-Bit unabhängig vom Wert des Filterbits automatisch akzeptiert.
  `MASK = 0x00000000` und `ID = 0x00000000` läßt alle IDs passieren.

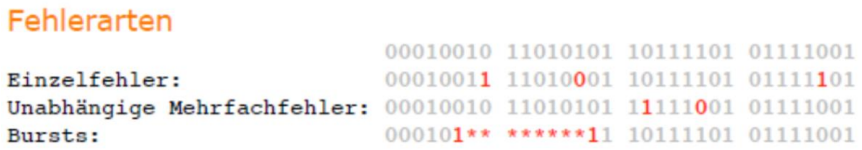
- Wenn ein Maskenbit auf Eins gesetzt ist, wird das entsprechende ID-Bit mit dem Wert des Filterbits verglichen. Wenn sie übereinstimmen, wird dies akzeptiert, andernfalls wird der Rahmen abgelehnt.

Beispiele:

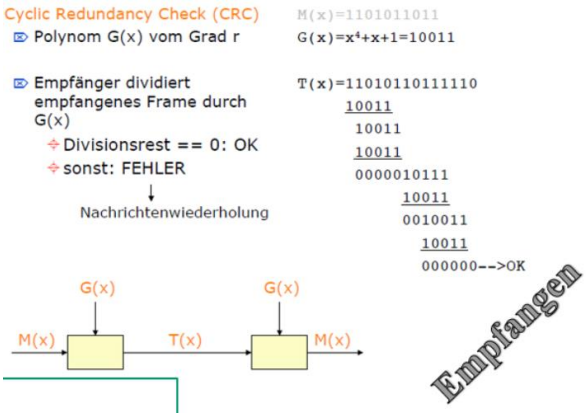
`MASK = 0x1FFFFFFF` und	`ID = 0x00001567`	für ID = 0x00001567
`MASK = 0x1FFFFFFF` und	`ID = 0x00000008`	für ID 8.
`MASK = 0x1FFFFFFE` und	`ID = 0x00000002`	für IDs 2 and 3.
`MASK = 0x1FFFFFF8` und	`ID = 0x00000000`	für IDs 0 to 7.
`MASK = 0x00000001` und	`ID = 0x00000001`	für alle ungeraden IDs.

11.11 CRC zur Fehlererkennung

CRC ist ein Verfahren um eine Anordnung mehrerer Prüfbits (Paritätsbits) zur Erkennung von Mehrfachfehlern und Burstfehlern zu finden, der mathematische Hintergrund ist vielfach in der einschlägigen Literatur zu finden.[7]
Welche Art von Fehlern gibt es:



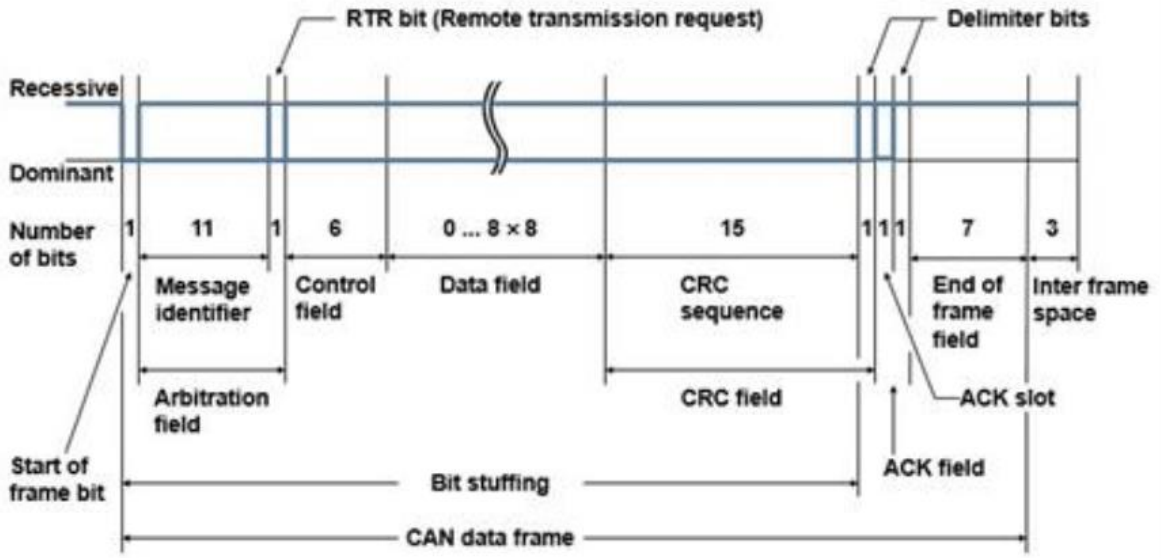
Ein Beispiel:[8]



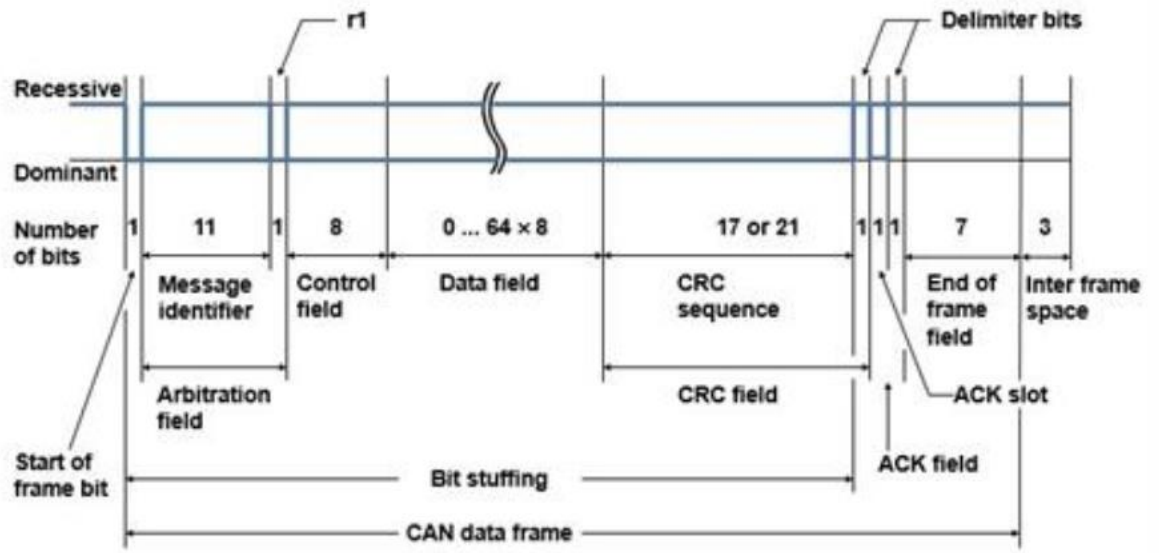
[7] https://www.inf.fh-flensburg.de/lang/algorithmen/code/crc/crc.htm
[8] Datenkommunikation, Hannes Federrath

11.12 CAN und CAN-FD Unterschied

Dies ist der normale CAN Rahmen mit einem 11 Bit Identifier:



Der CAN-FD Rahmen sieht ähnlich aus



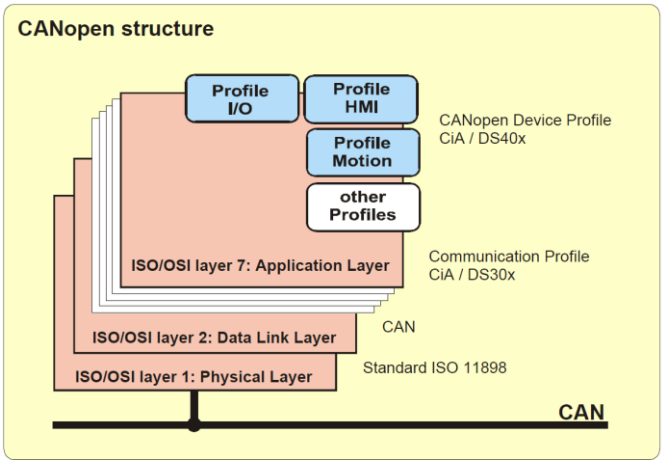
Die Nachrichten beginnen mit einem Start-Bit und gehen dann in den Nachrichten-Header mit dem Arbitrierungsteil über. Nach Abschluss der Arbitrierung beginnt der Nutzdaten-Teil. Am Ende wird die Nachricht durch einen CRC-Bereich und eine End-of-Frame-Sequenz abgeschlossen. Der Rahmen sieht ähnlich aus wie der des Standard CAN-Bus.

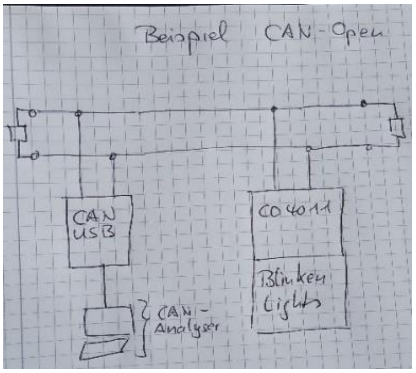
Unterschiede:

- Es gibt keinen Remote Frame, stattdessen ein reserviertes dominantes Bit (r1)
- 64 Byte statt 8 Byte im Datenfeld möglich: 0000 bis 1000 sind wie beim Standard CAN-Bus die Eintragungen für 0 bis 8 Byte Daten, in CAN FD werden die Werte 1001 bis 1111 für bis zu 64 Byte verwendet. Da es nicht genügend Belegungen gibt, um alle Längen zwischen 8 und 64 Byte zu repräsentieren, sind hier lediglich acht unterschiedliche Längen möglich.
- CRC Länge: Vorgabe ist die Hammingdistanz 6, deshalb unterschiedliche Längen. Ein CRC-15 dient wie gehabt für bis zu 8 Byte Daten, bis zu 16 Byte Daten werden durch einen CRC-17 und ein bis zu 64 Byte langes Datenfeld durch einen CRC-21 geschützt. Welcher CRC benötigt wird, wird am Data Length Code vor dem Datenfeld abgelesen. Im klassischen CAN wurden die Stuff Bits nicht durch den CRC abgesichert, was in seltenen Fällen zu Fehlern führen konnte.

12 CAN-Open[9]

CANopen ist ein auf den CAN-Bus basierendes Kommunikationsprotokoll, welches hauptsächlich in der Automatisierungstechnik z.B. für das Ansteuern von Motoren oder Stellgliedern und zur Vernetzung innerhalb komplexer Geräte verwendet wird.





12.1 Geräteprofile

CANopen beschreibt die Eigenschaften von Geräten in sogenannten Geräteprofilen. Diese sind im Prinzip definierte Variablensätze. Abhängig vom Gerätetyp werden bestimmte Daten bzw. Parameter (in CANopen als Objekte bezeichnet) fest definiert.

Geräteprofile (CiA)	für die Gerätetypen
DS 401	Digitale und analoge E/A
DS 402	Antriebe
DS 403	Bedienen und Beobachten
DS 404	Sensoren / Regler
DS 405	Programmierbare Geräte
DS 406	Encoder
DS ...	(weitere Geräteprofile)

Die Geräteprofile wurden von der CiA in den Standards DS40x beschrieben:

12.2 Objektverzeichnis

Das Objektverzeichnis ist die Zusammenstellung aller Variablen und Parameter (Objekte) eines CANopen Geräts. Dabei enthalten die Daten das Prozessabbild und mit den Parametern kann das Funktionsverhalten eines CANopen Gerätes beeinflusst werden. Ein Objektverzeichnis ist so aufgebaut, dass einige Parameter für alle Geräte

Object Index (hex)	Object
0000	Not used
0001 - 001F	Static Data Types
0020 - 003F	Complex Data Types
0040 - 005F	Manufacturer Specific Complex Data Types
0060 - 007F	Device Profile Specific Static Data Types
0080 - 009F	Device Profile Specific Complex Data Types
00A0 - 0FFF	Reserved for further use
1000 - 1FFF	Communication Profile Area
2000 - 5FFF	Manufacturer Specific Profile Area
6000 - 9FFF	Standardized Device Profile Area
A000 - FFFF	Reserved for further use

[9] CANOpen Guide Frenzel&Berg

dieser Kategorie zwingend vorgeschrieben sind und andere frei definiert und verwendet werden dürfen. Ein Beispiel zeigt der Ausschnitt aus dem Datenblatt des CO4011 Chips:

Das Schreiben auf einen digitalen Ausgang erfolgt über den Objektindex (oder auch Adresse genannt) 0x6200.

Index	6200h
Name	Write to digital output
Description	-
Data Type	Array

Index	Subindex 0
Name	
Description	Number of mapped objects
Data Type	Unsigned 8
Access modes	RO
PDO Mapping	NO
Value Range	-
Default Value	Number of digital output bytes

12.3 Telegrammaufbau

Aufbau eines Telegramms:

11 Bits	Byte 1	Byte 2 + 3	Byte 4	Byte 5 – Byte 8
FunctionCode+Adresse	Domain Protokoll	Objektindex	Subindex	Daten

Domain-Protokoll: In diesem Byte wird festgelegt was mit dem SDO bezweckt werden soll. Will man mit dem SDO etwas in ein Objekt schreiben, so muss dieses Byte den Wert 22h haben. Wenn ein Objekt ausgelesen werden soll, muss der Wert 40h sein.
40h lessen

12.4 Liste der Function-Codes

Identifier 11-Bit (binär)	Identifier (dezimal)	Identifier (hexadezimal)	Funktion
00000000000	0	0	Netzwerkmanagement
00010000000	128	80h	Synchronisation
0001xxxxxxxx	129 - 255	81h - FFh	Emergency
0011xxxxxxxx	385 - 511	181h - 1FFh	PDO1 (tx)
0100xxxxxxxx	513 - 639	201h - 27Fh	PDO1 (rx)
0101xxxxxxxx	641 - 767	281h - 2FFh	PDO2 (tx)
0110xxxxxxxx	769 - 895	301h - 37Fh	PDO2 (rx)
0111xxxxxxxx	897 - 1023	381h - 3FFh	PDO3 (tx)
1000xxxxxxxx	1025 - 1151	401h - 47Fh	PDO3 (rx)
1001xxxxxxxx	1153 - 1279	481h - 4FFh	PDO4 (tx)
1010xxxxxxxx	1281 - 1407	501h - 57Fh	PDO4 (rx)
1011xxxxxxxx	1409 - 1535	581h - 5FFh	SDO senden
1100xxxxxxxx	1537 - 1663	601h - 67Fh	SDO empfangen
1110xxxxxxxx	1793 - 1919	701h - 77Fh	NMT Error Controll
xxxxxxxx = Knotennummer 1 - 127			

12.5 Lesen der Vendor ID (CO4011):

Adresse des Knotens 0x40

Wie lautet die SDO Empfangs COBID?

0x640

Wie lautet der Domain Code?

0x40

Wie ist der Objekt Index + Subindex für Vendor ID?

0x1018 01

SDO Telegramm:
640 40 1810 01 00 00 00 00

Antwort:

5C0 42 1810 01 58 00 00 00

Die Vendor ID ist 0x00000058.

Object Dictionary

The CO4011 Single Chip CANopen Controller implements a complex object dictionary for CANopen I/O devices.

For detailed information about CANopen objects see additional brochure "Introduction to CANopen"

For the Object tables all values are shown in hexa-decimal way.
For access type the following settings are valid
ro read only
wo write only
rw read and write access enabled

DS301: global Objects

Index	Sub-Index	Name	Acc.
0005	-	Dummy 8	ro
0006	-	Dummy 16	ro
0007	-	Dummy 32	ro
1000	-	Device Type	ro
1001	-	Error Register	ro
1002	-	Manufacturer Status Register	ro
1005	-	COB-ID Sync Identifier Sync Object	rw
1008	-	Device Name *2)	rw
1009	-	Hardware Version *2)	rw
100B	-	Node Id	ro
100C	-	Guard Time	rw
100D	-	Life Time Factor	rw
100E	-	COB-ID Guard	rw
1010	-	Store Parameters *1)	wo
1011	-	Reload Default Parameter *1)	wo
1014	-	COB ID Emergency	rw
1015	-	Inhibit Time Emergency	rw
1017	-	Producer Heartbeat Time	rw
1018	0	Identity Object	ro
	1	Vendor ID	ro
	2	Product Code	ro
	3	Revision Number	ro

12.6 Lesen der Konfiguration

```
5        146.066.287.8        S      601    8              40 01 21 00 00 00 00 00
6        146.069.126.7               581    8              42 01 21 00 04 00 00 00
```

Welche Adresse hat der Knoten?
Was ist Sende- und was ist Empfangs-SDO?
Wie lautet der Objektindex?
Wie lautet der Domaincode für die Anfrage und wie für die Antwort?
Welcher Subindex?
Wie ist die Antwort?

Einige Beispiele:

Für welche Knotenadresse gelten diese Kommandos?

703	***00***	*Boot up Message vom Knoten 3*
0	***01 03***	*Start Node vom Master an den Knoten 3*
183	***01 00 00***	*Transmit PDO1 als Antwort vom Knoten auf die Boot up Message*
603	***22 00 62 01 55 00 00 00***	*Empfangs SDO mit schreiben auf Objekt 6200 Subindex 1 den Wert 55h*
583	***60 00 62 01 00 00 00 00***	*Sende SDO mit Bestätigung, dass Schreiben auf Objekt 6200 erfolgreich war*
603	***40 00 10 00 00 00 00 00***	*Empfangs SDO Leseanforderung Objekt 1000*
583	***42 00 10 00 91 01 03 00***	*Sende SDO mit Bestätigung und Inhalt Objekt 1000: 0003 0191h (Unsigned 32)*
203	***23 89***	*Empfangs PDO1: der Wert 23h wird laut Mapping ins Objekt 6200 Sub. 01 geschrieben, der Wert 89h ins Objekt 6200 Sub. 02*
183	***41 00 00***	*Sende PDO: Laut Mapping hat das Objekt 6000 Sub. 01 den Wert 41h (Digitales Eingangsbyte 0)*
183	***01 00 00***	*Sende PDO: Laut Mapping hat das Objekt 6000 Sub. 01 den Wert 01h (Digitales Eingangsbyte 0)*

Die oben gezeigte Sequenz stammt aus dem Frenzel&Berg canopenstarter Manual.

Der folgende kurze Ausschnitt ist aus dem Objektverzeichnis eines Schrittmotors von Berger & Lahr:

Objekt	Bedeutung
profile velocity (6081h)	Endgeschwindigkeit
profile acceleration (6083h)	Beschleunigungsrampe
profile deceleration (6084h)	Verzögerungsrampe

Die Rampenwerte können nicht remanent gespeichert werden.

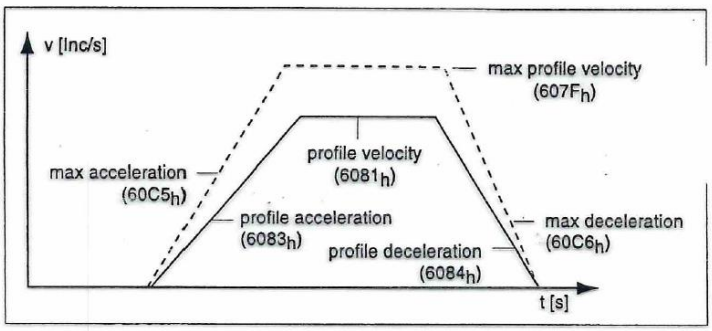
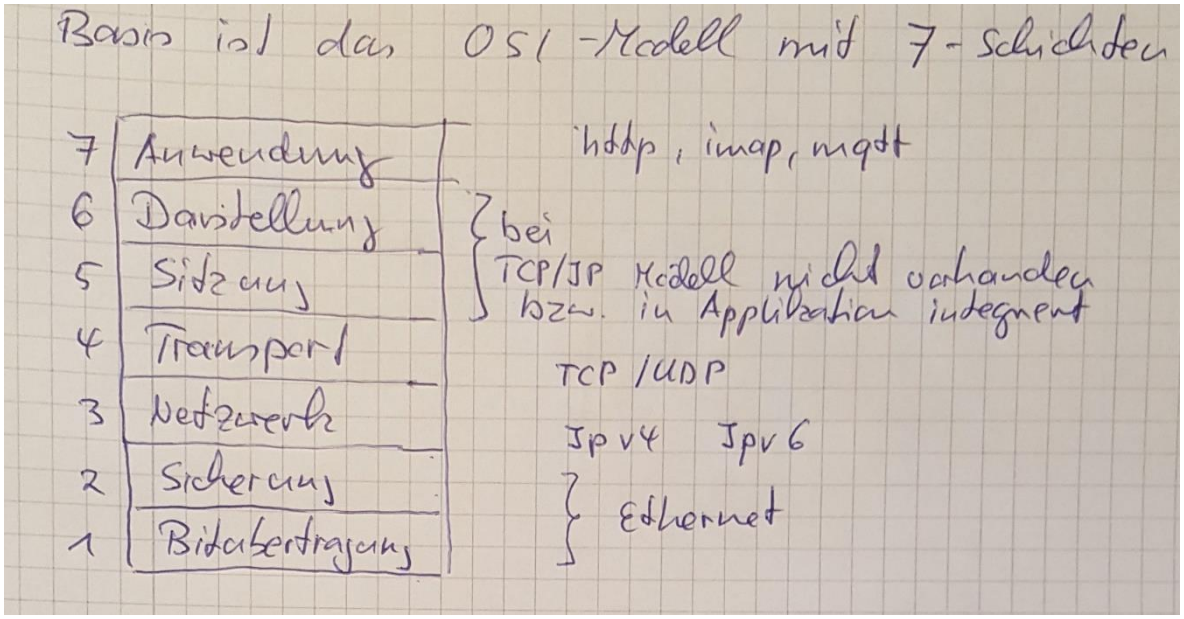


Bild 5.11 Rampeneinstellungen

13 TCP/IP Schichtenmodell

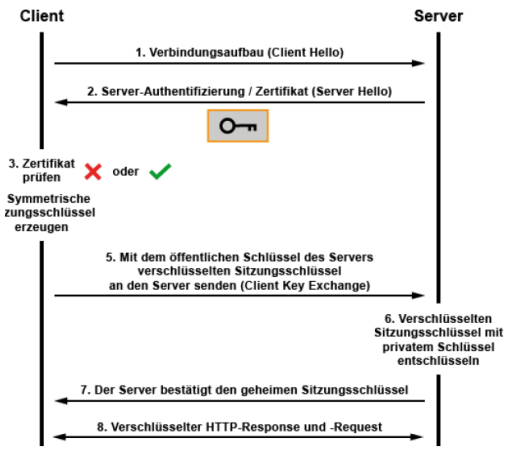


Darstellung:
Interpretiert die Daten der Anwendung, Codierung und Dekodierung
(Oft in der Anwendung enthalten oder fehlt gänzlich).

Sitzung:
Aufbau, Durchführung und Beenden der Sitzung

14 HTTP/HTTPS

Wie unterscheiden sich HTTP und HTTPS? Die einfache Antwort lautet: technisch betrachtet gar nicht! Das Protokoll selbst, also die Syntax, ist bei beiden Varianten identisch.



Der Unterschied liegt im Transportprotokoll, das herkömmliche http nutzt TCP als Transportprotokoll, https nutzt eine weitere Zwischenschicht im Transportprotokoll mit der Bezeichnung SSL/TLS. Nicht das Protokoll selbst, sondern dessen Übertragungsart ist zusätzlich gesichert. Dazu folgende Analogie:

- Zwei Personen telefonieren miteinander.
- Sie benutzen eine gemeinsame Sprache, um sich zu verständigen: HTTP.

- Die Telefonverbindung, über die ihr Gespräch läuft, ist im Falle von HTTP ungesichert und im Falle von HTTPS auf besondere Weise abhörsicher.

Die folgende Tabelle fasst die wichtigsten Unterschiede aus Benutzersicht zusammen:

	HTTP	HTTPS
Übertragung	Unverschlüsselt	Verschlüsselt
Zertifikat	Nein	Ja
Port-Nummer	80	443
Adressierung in der URL	http://	https://

Kennzeichnung des Sicherheitsstandards in der Adresszeile verschiedener Browser

Browser	http://	https://
Chrome	Nicht sicher example.com	example.com
Firefox	example.com	https://example.com
Opera	Not secure example.com	example.com
Microsoft Edge	example.com/	https://example.com/

IONOS

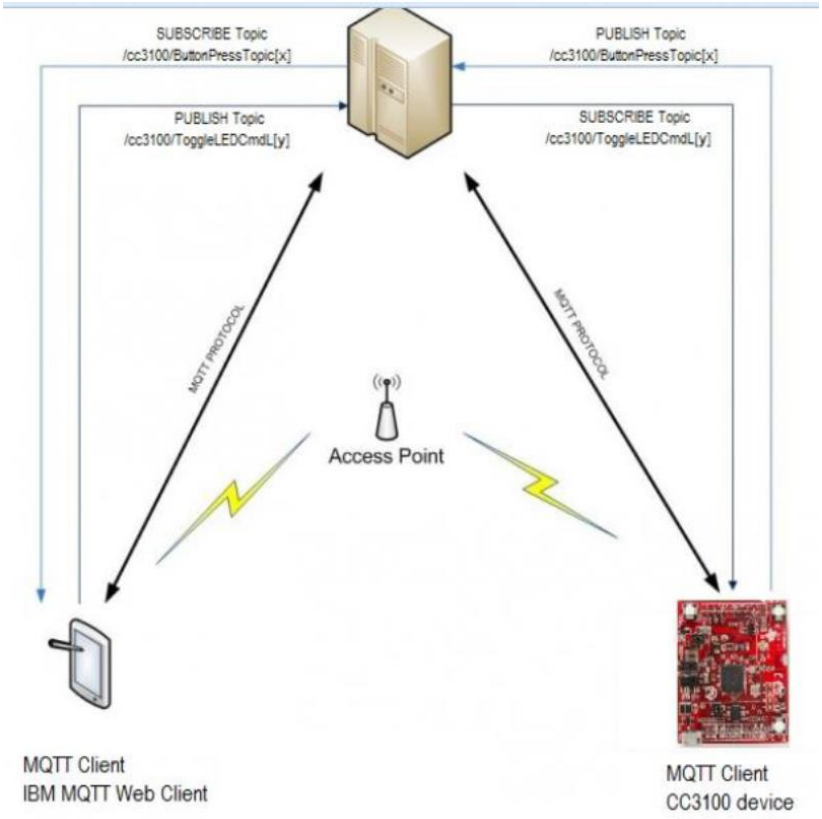
15 MQTT Protokoll

Das *MQ Telemetry Transport* ist ein schlankes Messaging Protokoll, welches auf den Bereich Mobile und *Internet of Things* zugeschnitten ist. MQTT wurde von Andy Stanford-Clark (IBM) und Arlen Nipper (Eurotech; jetzt Cirrus Link) im Jahr 1999 für die Überwachung einer Öl-Pipeline durch die Wüste entwickelt. Für die Anwendung wurde ein Protokoll benötigt das die Bandbreite effizient ausnutzen konnte und eine geringe Batteriekapazität benötigte. Die Komponenten waren über einen Satelliten-Link vernetzt und dies war zu dieser Zeit extrem teuer. Die Übertragung von Nachrichten erfolgt dank des kompakten Binärprotokolls annähernd in Echtzeit. Über den Inhalt einer Nachricht macht MQTT keine Annahmen. Text-, Binär- oder Objektnachrichten sind möglich. Inzwischen ist MQTT ein OASIS Standard und liegt aktuell in der Version 3.1.1 vor. Zentrale Komponente ist ein `Broker` (in der Cloud), z.B.

- Host mit Netzwerkanschluss und „Broker-Software“, z.B. ***Mosquitto*** (Open Source)
- Dedizierter Broker einer Firma

Grundlagen

- Eine `message` hat eine `topic` und eine `payload`, entspricht dem Betreff und Inhalt einer EMail.
- Der `Publisher` schickt die Nachricht an ein Netzwerk.
- The Subscriber hört auf Nachrichten mit einem bestimmten topic.
- Der `Broker` ist verantwortlich für die Koordination der Kommunikation zwischen `Publisher` und `Subscriber`. Ein Broker kann beispielsweise Nachrichten speichern wenn die Subscriber offline sind.
- Das Protokol verwendet eine publish/subscribe Architektur im Gegensatz zu HTTP mit dem request/response Paradigma.



15.1 Quality of Service

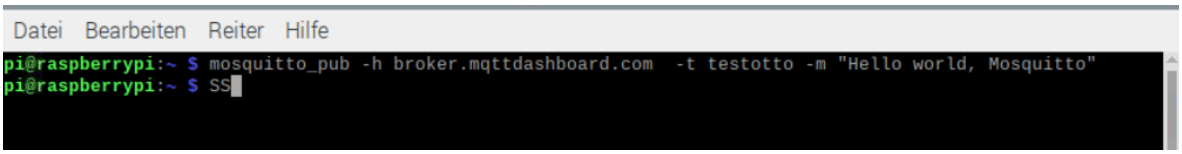
Wie oft werden die Daten zugestellt? Diese Frage beantwortet das ***Quality of Service*** (QoS) im Header des Protokolls:

QoS steht für den Grad der Zuverlässigkeit, mit der Nachrichten zugestellt werden. MQTT kennt die drei QoS Stufen 0, 1 und 2. Jede Stufe steht für bestimmte Garantien, auf deren Einhaltung man sich verlassen kann. Jede Stufe stellt einen Kompromiss zwischen Zuverlässigkeit und Ressourcenverbrauch dar. Bei Homeautomation, Sensoren oder Internet of Things kann der Ressourcenverbrauch wichtiger sein als die Zuverlässigkeit. Der Sender kann bei jeder Nachricht individuell die QoS Stufe angeben:

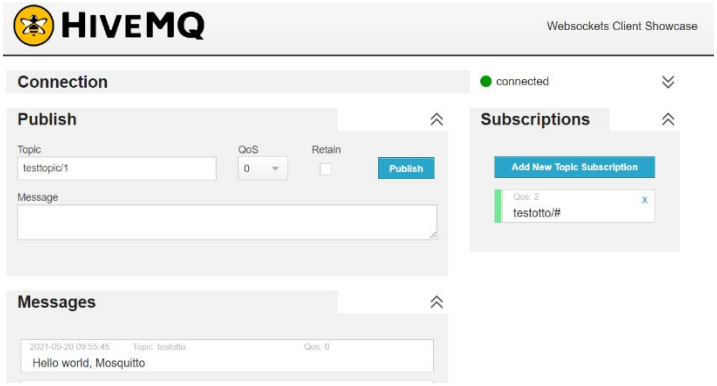
QoS	Name	Beschreibung	Nachrichten-Verlust ist möglich	Duplikate sind möglich
0	At most once	Nachricht wird genau einmal verschickt. Es gibt keine Antwort und keine Bestätigung vom Server. Ein Retry wird nicht durchgeführt. Schnellste Art des Nachrichtenversandes	ja	nein
1	At least once	Nachricht kann mehrfach verschickt werden. Server antwortet mit einer Bestätigung.	nein	ja
2	Exactly once	Erhöhter Overhead	nein	nein

15.2 Beispiel Raspberrypi/mosquitto

Als Beispiel wird eine Kommunikation zwischen einem RaspberryPi Mqtt Client und dem Broker HiveMQ gezeigt. Auf dem RaspberryPi wird eine Nachricht gesendet („publish“).

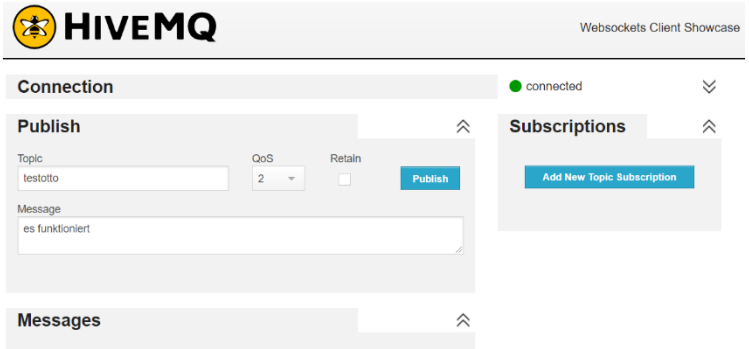


Der Websocketclient zeigt dann diese an, nachdem zu dem Topic „testotto“ subscribed wurde:



In der umgekehrten Richtung funktioniert dieser Prozess selbstverständlich auch:

Im Websocket Client wird gesendet („publish“)



Der RaspberryPi erhält die Nachricht, nachdem er sich zu dem Topic „testotto“ subscribed hat

```
pi@raspberrypi:~ $ mosquitto_sub -h broker.mqttdashboard.com  -v -t testotto
testotto es funktioniert
```

16 Anhang ASCII-Tabelle

ASCII ist eine Abkürzung für (American Standard Code for Information Interchange). Dieser Standard hat sich bis heute bewährt. Dieser Standard definiert 128 Zeichen, die auf den Computern der westlichen Welt am häufigsten gebraucht werden. Dies beinhaltet beispielsweise alle Buchstaben des lateinischen Alphabeths, alle Ziffern und weitere Sonder- und Steuerzeichen.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

www.VirtualUniversity.ch