Mikroprozessortechnik

Prof. Dr. Michael Lipp

fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

USART: Universal Synchronous and Asynchronous Serial Receiver and Transmitter

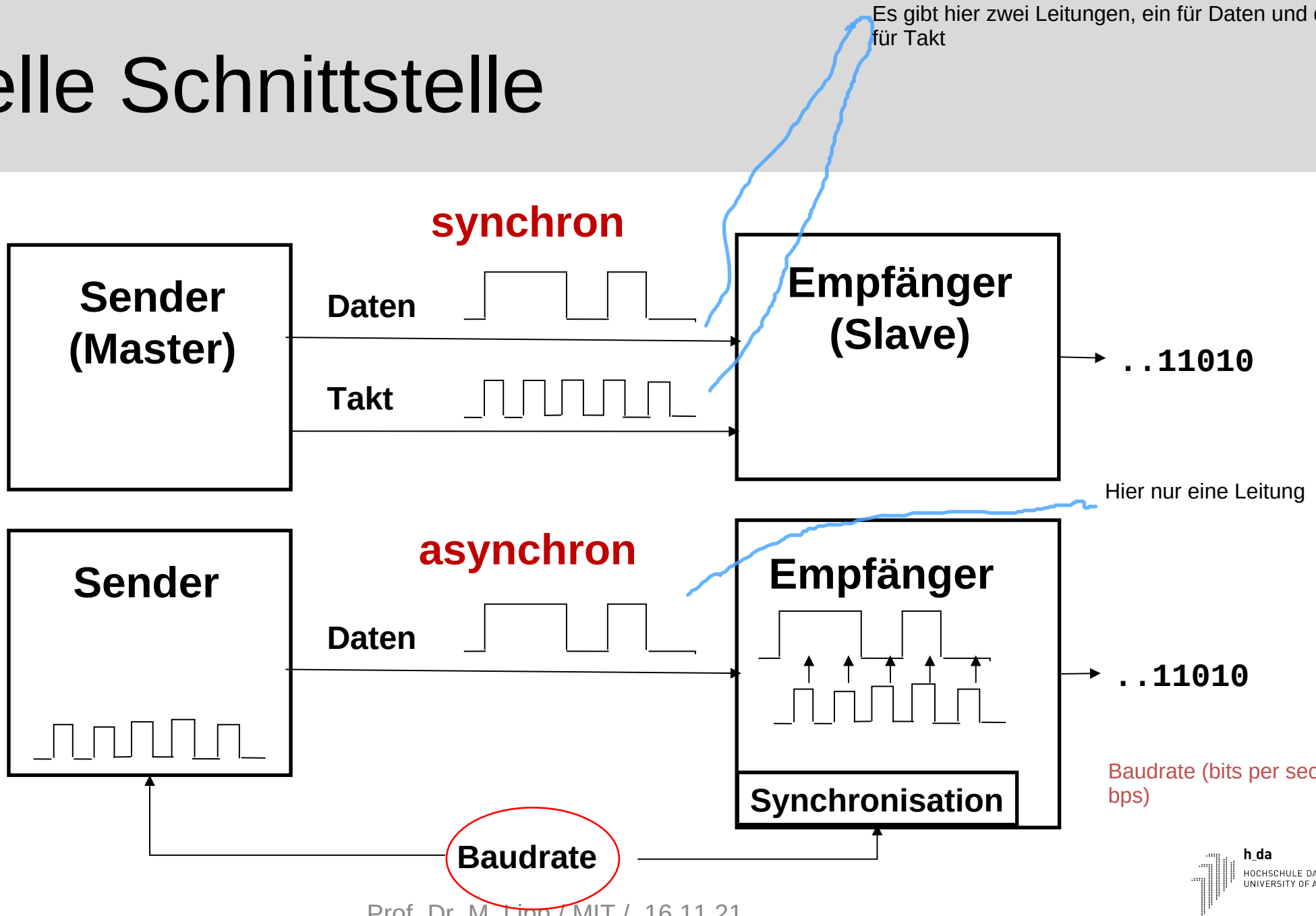
Serielle Schnittstelle

Serielle Schnittstelle

- **Zeitlich hintereinander folgende Übertragung einzelner Bits**
  - ein Bit wird pro Takt empfangen/gesendet
  - Asynchron oder synchron
  - Beispiele:

UART	Universal Asynchronous Receiver and Transmitter	asynchron
USART (RS232, V.24)	Universal Synchronous and Asynchronous Receiver and Transmitter	asynchron synchron
SPI	Serial Peripheral Interface	synchron (peer-to-peer)
TWI / I^2C	Two Wire Interface	synchron (Bus)

Serielle Schnittstelle



Asynchron bedeutet nicht das der Daten willkürlich übertragen werden

Asynchron bedeutet lediglich das kein Takt übertragen wird.

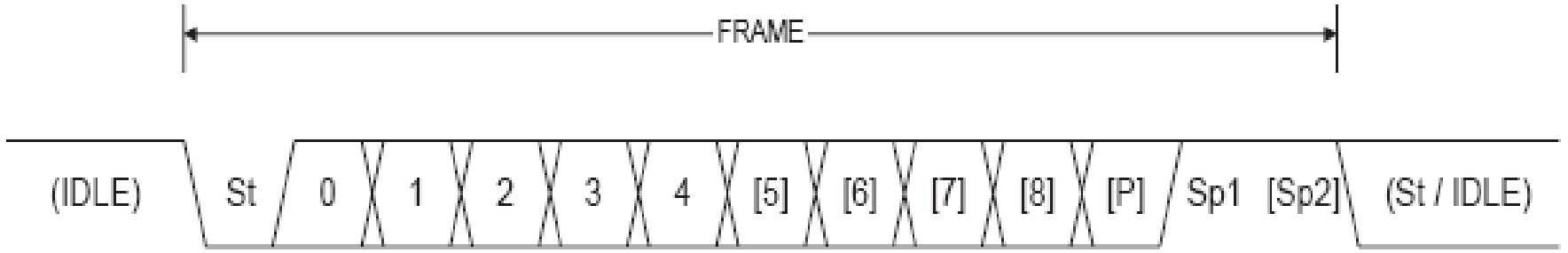
Baudrate (bits per second oder bps)

Serielle Schnittstelle

The number of data and formatting bits, the order of data bits, the presence or absence of a parity bit, the form of parity (even or odd) and the transmission speed must be pre-agreed by the communicating parties. The "stop bit" is actually a "stop period"; the stop period of the transmitter may be arbitrarily long. It cannot be shorter than a specified amount, usually 1 to 2 bit times. The receiver requires a shorter stop period than the transmitter. At the end of each character, the receiver stops briefly to wait for the next start bit. It is this difference which keeps the transmitter and receiver synchronized.

- **Data Frame**

Der Empfänger sieht nur das auf der Daten Leitung aus



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

Bei Start low d.h. ein fallende Flanke

Nach der Start kommt die Data Bits mit der niederwertige Bit

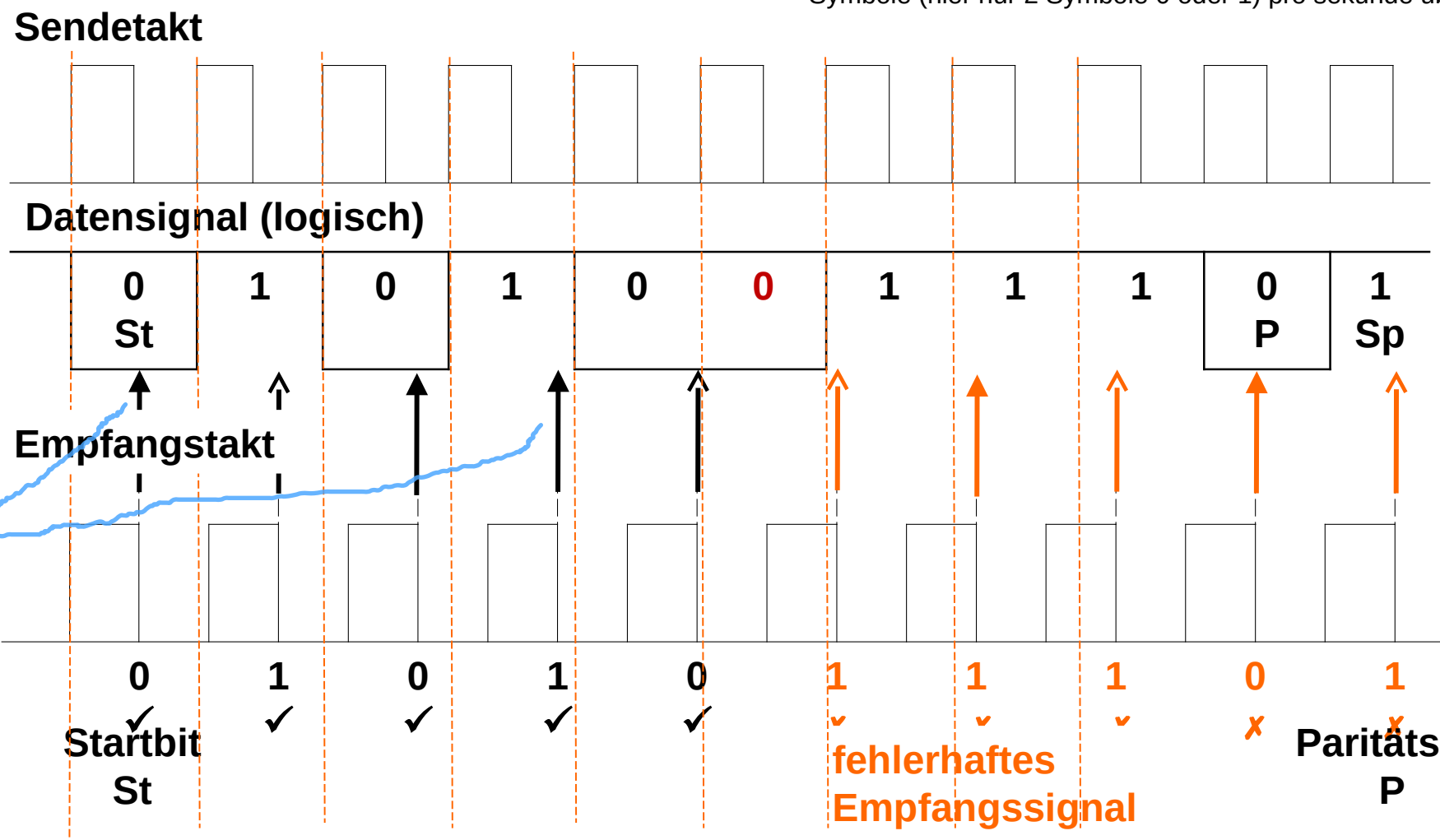
Der Parity bit sagt ob die vorher gezählte eins waren gerade oder ungerade

Wenn nichts übertragen wird, dann ist er in IDLE Form

Serielle Schnittstelle

Ich bekomme hier kein Takt über der Leitung übertragen sondern ich habe hier meine eigene Taktgeber

Der Sender und der Empfänger müssen auf eine Baudraht einigen d.h wie viele Symbole (hier nur 2 Symbole 0 oder 1) pro sekunde übertragen werden.



Beim Start gibts immer ein fallende Flanke

Es verschiebt sich langsam von der mitte zum hinten

Ich kann jeder steigende oder fallende Flanke des Datensignal eigentlich zur Synchronisation nutzen

Wenn die steigende oder fallende Flanke des Datensignal nicht zeitlich mit der steigende Flanke von meinem Taktgebers geht dann ist etwas faul

Worst Case - ALLE Daten Bits nur 0 oder 1 sind. Das ich mindesten jeder 10 Taktperiod ein Wechsel habe

Empfangstakt weicht vom Sendetakt ab

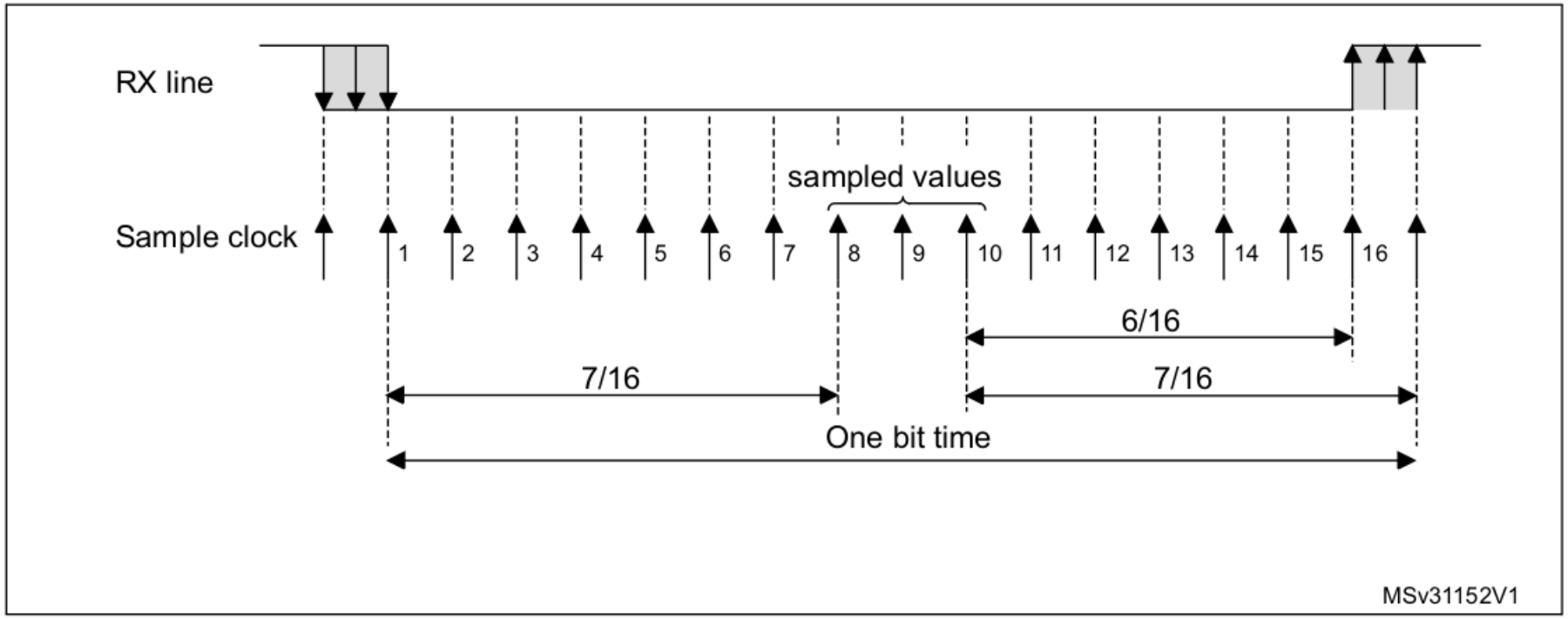
Deshalb ist in Algorithmus so eingebaut, das bei der Abweichungen es gibt Automatische Korrektur

Serielle Schnittstelle

- **Taktsynchronisation (Clock Recovery) durch Oversampling**
  - Detektierte Flanken „justieren“ den schnelleren Abtasttakt

16 mal schneller läuft als der eigentliche Empfangstakt

Figure 172. Data sampling when oversampling by 16



RS-232 – das Original

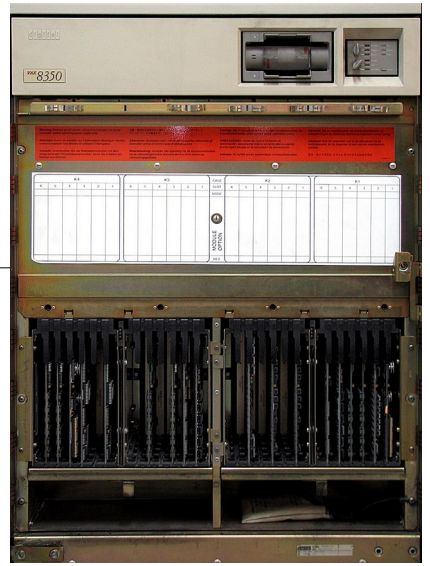
Modem

RS-232

Telefonnetz

Modem

RS-232

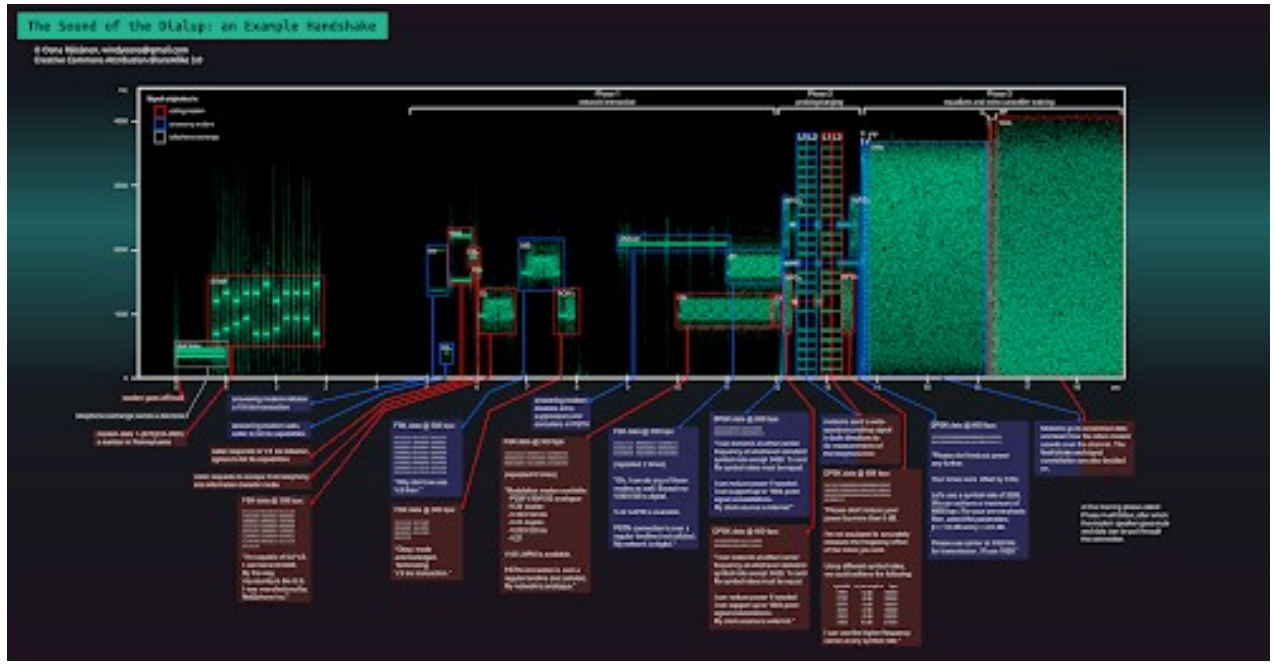


DTE (DEE)	**D**ata **T**erminal **E**quipment (Daten-Endeinrichtung)
DCE (DÜE)	**D**ata **C**ommunication **E**quipment (Datenübertragungseinrichtung)
TXD	**T**ransmit e**X**change **D**ata
RXD	**R**eceive e**X**change **D**ata

Bildquellen: Wikipedia

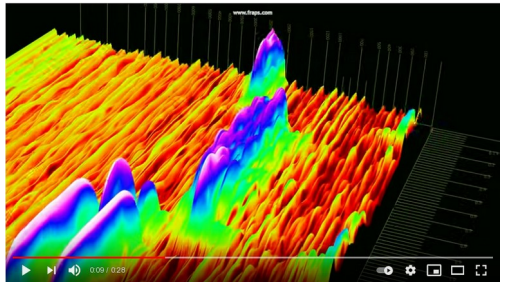
RS-232 – das Original

- **Verbindungsaufbau Modem**



Quelle:
http://www.windytan.com/2012/11/the-sound-of-dialup-pictured.html

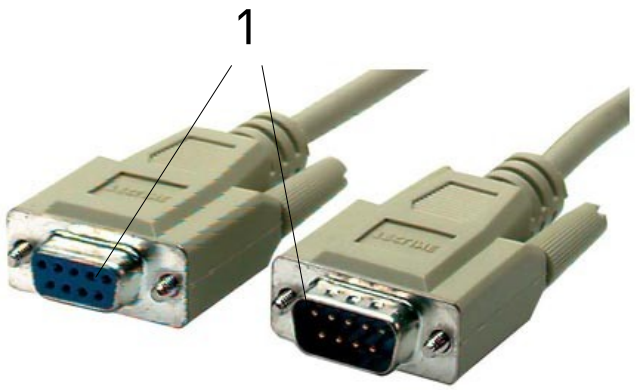
Weitere schöne Darstellung:
https://www.youtube.com/watch?v=vvr9AMWEU-c

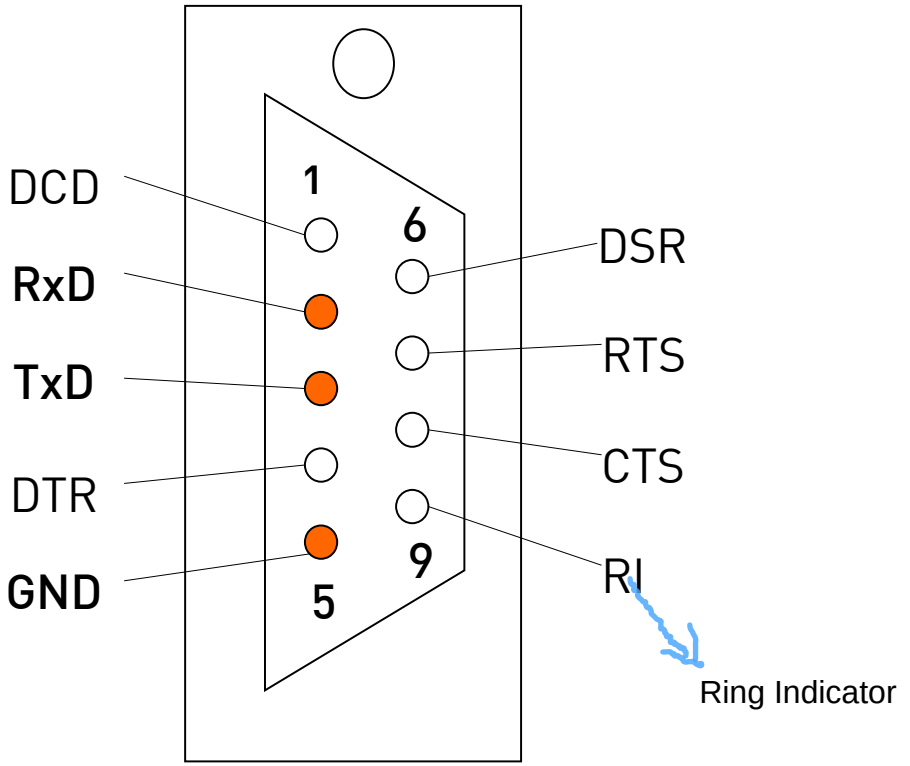


- **(Grundsätzliches Prinzip bis heute unverändert https://www.youtube.com/watch?v=7m8pAuk9lsk)**

RS-232

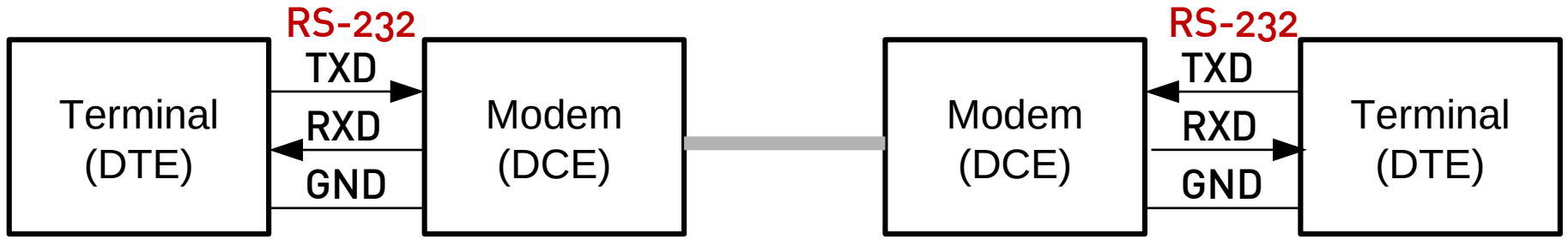
- **Physikalische Belegung**
  - Fett: minimale Belegung
  - Zusätzliche Handshake- und Modem-Signale

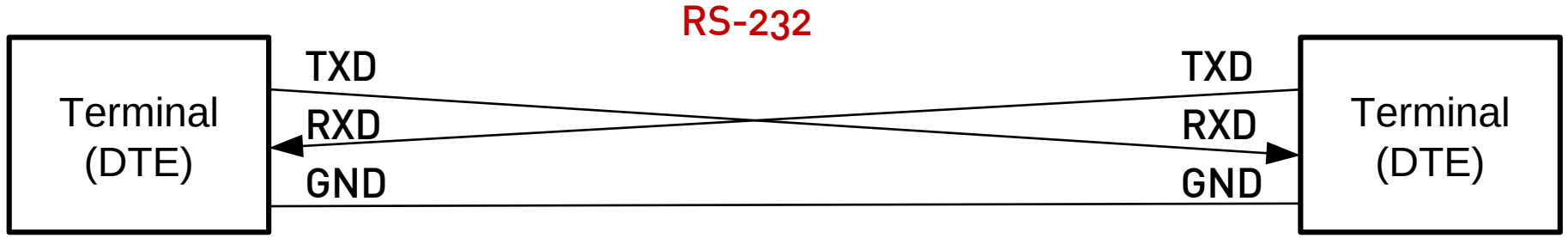




RS-232

- **Direkte Verbindung zweier DTEs**





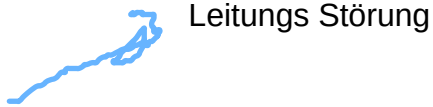
„Null-Modem“ Kabel
(TXD/RXD gekreuzt)

kein Modem benutzt

RS-232

Aufgefallen ist vielleicht, dass bei der Übertragung eine 0 auf der Datenleitung als 1-Signal erscheint. Das hängt damit zusammen, dass die RS232-Schnittstelle nach Vereinbarung eine invertierende Spannungsschnittstelle (negative Logik) ist. Spannungspegel zwischen -3V und -15V entsprechen einer logischen 1, Spannungspegel zwischen +3V und +15V einer logischen 0. Alle Signalpegel zwischen -3V und +3V sind undefiniert.

- **Physikalische Darstellung RS-232**
  - Negative Logik
  - Pegel zwischen -3 V und +3 V sind undefiniert
  - Maximale Leitungslänge bei 2.400 Bd: 900 m

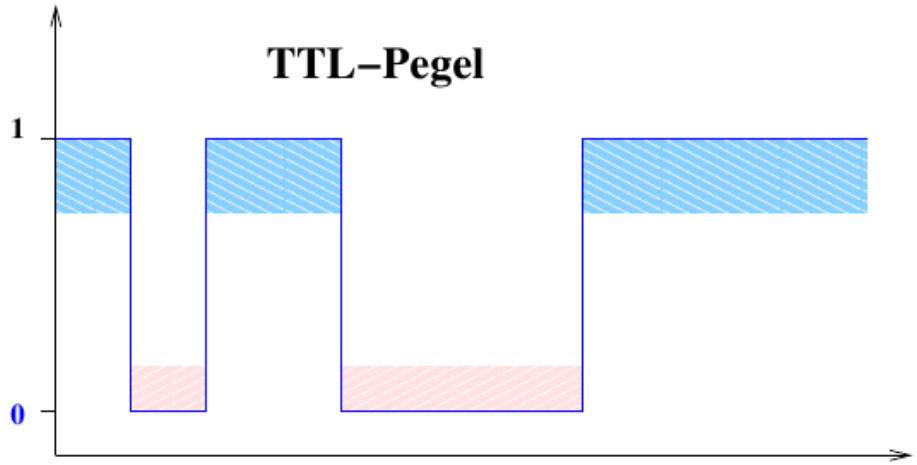


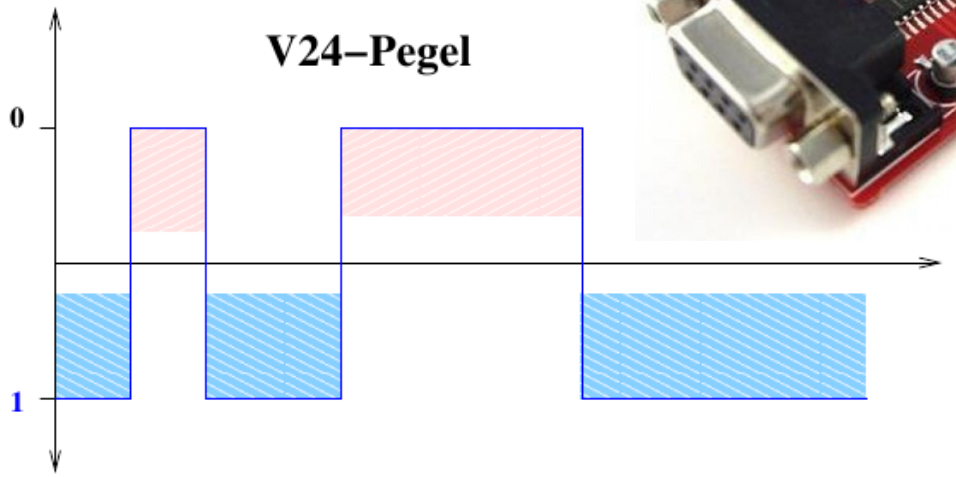
Bit	Sender-Pegel [V]			Empfänger-Pegel [V]		
	min.	norm.	max.	min.	norm.	max.
0	+5	+12	+15	+3	+12	+15
1	-5	-12	-15	-3	-12	-15

Serielle Schnittstelle

Man kan gleiche Protokoll mit TTL Pegeln machen. Hier braucht man ein Pegel Wandler

- **Physikalische Darstellung mit TTL-Pegeln**
  - USART-Schnittstellen von (Evaluation-)Boards verwenden oft TTL-Pegel
  - Bei Bedarf Anpassung mit Pegelwandlern





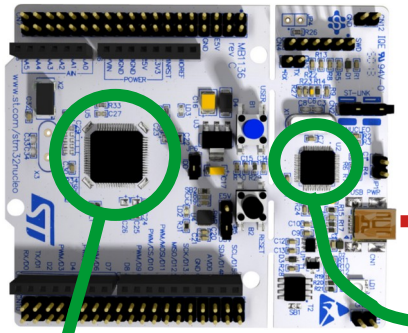
Sicherer Pegel ’1’

Sicherer Pegel ’0’

Serielle Schnittstelle Nucleo-Board

- **„Tunnelung“ durch USB-Verbindung des Programierteils**
  - Am PC als „Virtual COM Port“ verfügbar
  - Simulation des Terminals auf dem PC durch das Programm PuTTY

COM Port ist ein Communication Port

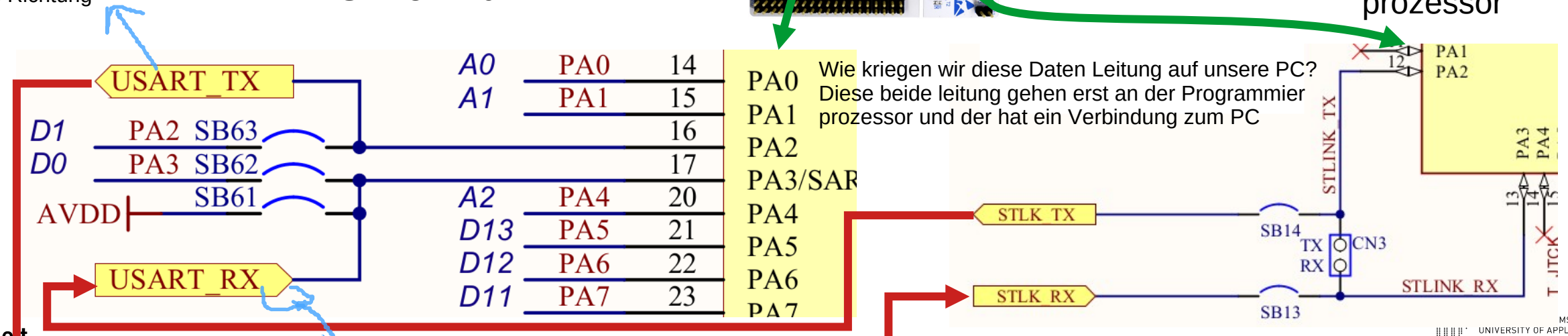


PC

Programmier-
prozessor

Das ist der Transmit Leitung in der sende Richtung

STM32F401RE



Wie kriegen wir diese Daten Leitung auf unsere PC? Diese beide leitung gehen erst an der Programmier prozessor und der hat ein Verbindung zum PC

Das ist der Receive Leitung in der Empfangs Richtung

Serielle Schnittstelle

- **Was können wir senden?**
  - 5, 6, 7, 8, (9) bit-Werte → Typisch: 8 bit = 1 Byte
  - Bei Verbindung mit einem Terminal: Schriftzeichen

Einschub: Schriftzeichen

- **Rechner kann nur Werte Speichern**
- **Speicherung von Schriftzeichen erfolgt durch Zuordnung von Werten zu Schriftzeichen**
  - → Zeichenkodierung
- **Beispiele**
  - ASCII
  - ISO-8859-15
  - UTF-8

Einschub: Schriftzeichen

- **ASCII (American Standard Code for Information Interchange)**

ASCII-Zeichentabelle, hexadezimale Nummerierung

Code	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
0…	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1…	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2…	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3…	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4…	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5…	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6…	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7…	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Quelle: Wikipedia

Einschub: Schriftzeichen

- **ASCII**
  - 7-bit Code
  - Werte 0x00 bis 0x1F und 0xFF sind Steuerzeichen
    - Beispiele:
      - 0x0D → Wagenrücklauf (an Anfang der Zeile) (engl. Carriage Return, CR)
      - 0x0A → Zeilenvorschub (Nächste Zeile) (engl. Line Feed, LF)
      - 0x07 → Klingel/Glocke (Bei Schreibmaschinen kurz vor Zeilenende ausgelöst) (engl. Bell, BEL)
      - 0x08 → Rückwärts(leer)zeichen (engl. Backspace, BS)
      - 0x7F → Löschen (engl. Delete, DEL)

„Fernschreiber T100 Siemens“ von Nightflyer/CC BY-SA

Z.B. https://www.youtube.com/watch?v=FnKQKpq6Kt8

Einschub: Schriftzeichen

- **ASCII**
  - „Eingebaut“ in C/C++
    - **Niemals** ASCII-Codes (Werte) in Programmen verwenden!
    - Einfache Anführungszeichen repräsentieren die ASCII-Codes
      - `'A'` → Wert, der in der ASCII-Zeichenkodierung dem Buchstaben „A“ zugeordnet ist
      - Notation für Steuerzeichen: `'\r'` (CR), `'\n'` (LF) (allg. Oktalzahl: `'\`*nnn*`'`)
  - Explizite Nutzung von Wissen bezüglich der Kodierung nur in speziellen Fällen
    - `char c = '5'; int value = c – '0';`

Einschub: Schriftzeichen

- **ISO-8859-1 (-15)**
  - 8-bit Code
  - „Latin-1“
  - Verwendet die Werte 128-255 um möglichst viele Zeichen westeuropäischer Sprachen zu repräsentieren
  - Weitgehende Übereinstimmung mit Windows-spezifischem Zeichensatz CP-1252
  - Variante ISO-8859-15 enthält Euro-Zeichen

Einschub: Schriftzeichen

- **UTF-8 (8-Bit UCS Transformation Format)**
  - Komprimierte Kodierung des (32-Bit) Universal Coded Character Set (UCS)
  - Werte 0-127 identisch zu ASCII

32 bit = 4 Bytes

Unicode-Bereich (hexadezimal)	UTF-8-Kodierung (binär, Schema)	Algorithmus/Erläuterungen	Anzahl der codierbaren Zeichen	
0000 0000 – 0000 007F	0xxxxxxx	In diesem Bereich (128 Zeichen) entspricht UTF-8 genau dem ASCII-Code: Das höchste Bit ist *0*, die restliche 7-Bit-Kombination ist das ASCII-Zeichen.	2^7	128
0000 0080 – 0000 07FF	110xxxxx 10xxxxxx	Das erste Byte beginnt immer mit 11, die folgenden Bytes mit 10. Die xxxxx stehen für die Bits des Unicode-Zeichenwerts. Dabei wird das niederwertigste Bit des Zeichenwerts auf das rechte x im letzten Byte abgebildet, die höherwertigen Bits fortschreitend *von rechts nach links*. Die Anzahl der Einsen vor der ersten 0 im ersten Byte ist gleich der Gesamtzahl der Bytes für das Zeichen. (*Rechts* in Klammern jeweils die theoretisch maximal mögliche Zahl kodierbarer Zeichen, die aber aufgrund von Einschränkungen im Unicode- oder UTF-8-Standard nicht in vollem Umfang verwendet werden dürfen.)	$2^{11} - 2^7$ (2^{11})	1920 (2048)
0000 0800 – 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx		$2^{16} - 2^{11}$ (2^{16})	63.488 (65.536)
0001 0000 – 0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx		2^{20} (2^{21})	1.048.576 (2.097.152)

Die nummer von eins sagen, wie viele Bytes hinter her kommen

Quelle: Wikipedia

Einschub: Schriftzeichen

- **UTF-8**
  - Beispiele
    - ä → 0xc3 0xa4
    - ö → 0xc3 0xb6
    - ü → 0xc3 0xbc
    - € → 0xe2 0x82 0xac

binär 10000000 – 10111111
Hexadezimal 80 – BF
Decimal range128 – 191
Zweites, drittes oder viertes Byte einer Bytesequenz
They always start from 10 in the beginning

binär 00000000 – 01111111
Hexadecimal 00 – 7F
decimal range 0 – 127
Ein Byte lange Zeichen, deckungsgleich mit US-ASCII

110xxxxx- Start of 2 Bytes Sequenz
1110xxxx- Start of 3 Bytes Sequenz
11110xxx-Start of 4 Bytes Sequenz

Note - Not more than 4 Bytes, weil Beim UTF-8 Code werden alle Zeichen mit minimal 1 und maximal 4 Byte kodiert.

Einschub: Schriftzeichen

- **Zeichenketten (Strings)**
  - Repräsentation in C
    - `'\0'`-terminierte Folge von `char`
    - Länge berechnet durch „Suchen“ des Bytes mit Wert 0
    - Wird typischer Weise auf dem Heap gespeichert
    - Hilfsfunktionen `strlen`, `strncpy`, `strdup`, …
  - Repräsentation in C++
    - Klasse `string`
    - „Dynamisches Array von char“
    - Konvertierung zu C-String mit Methode `c_str()`
    - Konvertierung von C-String nach `string` mit Konstruktor `string(char*)`

„`char`“ ist ein eigener Typ für Schriftzeichen und nicht identisch zu „`signed char`“ (wie es bei numerischen Typen der Fall ist)

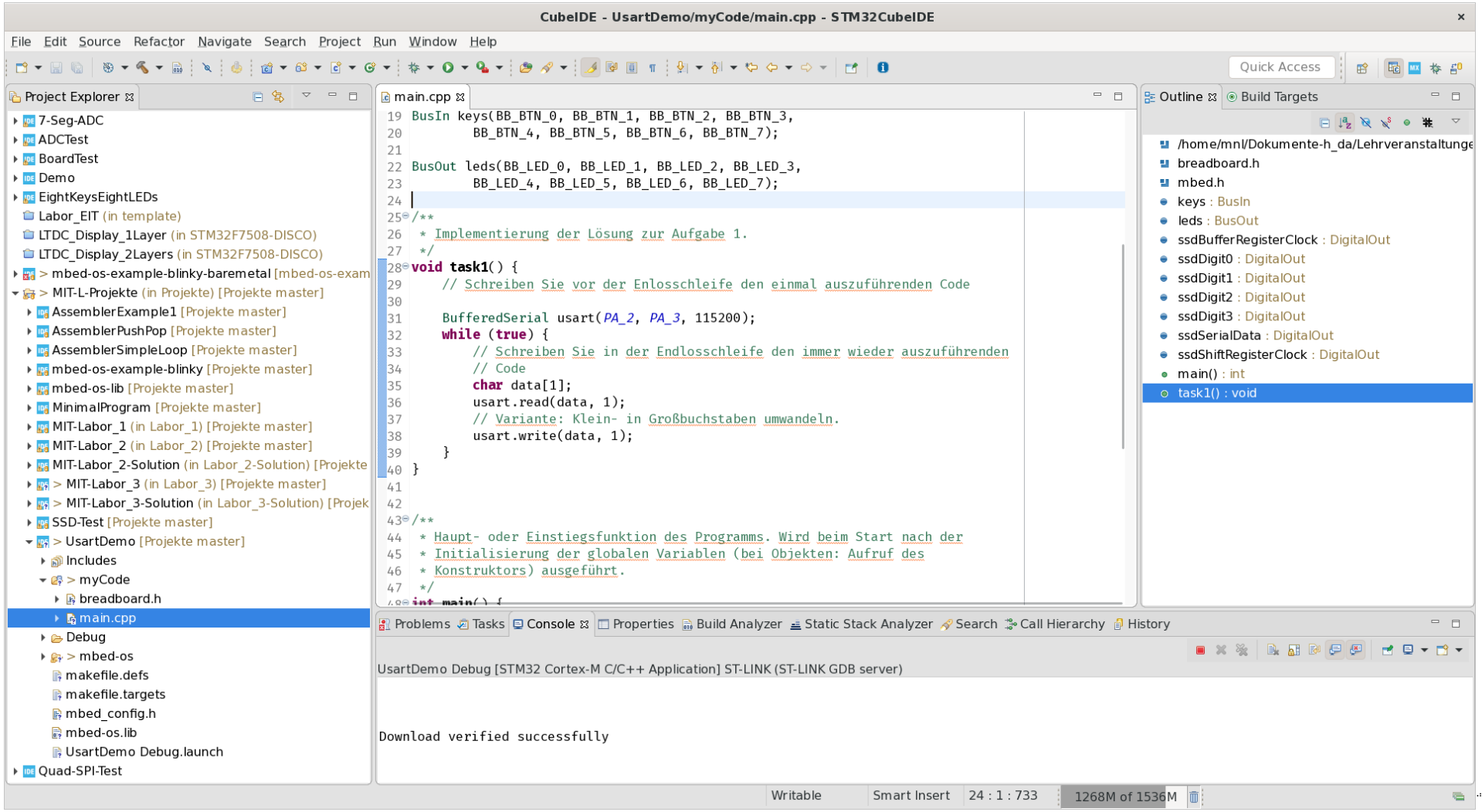
Generell sollte aufgrund des begrenzen Speichers der Heap beim Programmieren von Embedded Systemen immer mit Vorsicht und sorgfältiger Abschätzung der Auslastung genutzt werden!

Serielle Schnittstelle

- **Modellierung in Mbed mit Klasse `BufferedSerial`**
  - Angabe von Pin für Senden (TX), Pin für Empfangen (RX) und Baud Rate als Argumente des Konstruktors
  - Schreiben von Daten mit
    `ssize_t write(const void* buffer, size_t length)`
  - Lesen mit
    `ssize_t read(const void* buffer, size_t length)`
    - Blockiert (d. h. wartet), wenn keine Daten vorhanden sind
    - Verhalten kann geändert werden mit `int set_blocking(bool blocking)`

Live Coding

Serielle Schnittstelle



Diese Programm implementiert ein Echo Es liest ein Zeichen in diese ein Zeichen große puffer und dann gibt es genau ein Zeichen aus dem puffer wieder aus. Das heißt jeder Charakter den wir auf dem PC eintippen , der wird wieder an der PC zurück geechot

C/C++-Wissen: to/from string

- **C**
  - `int atoi (const char * str);` atoi - ASCII to Integer
  - `long int atol ( const char * str );`
  - `double atof (const char* str);` Der Rückgabewert ist hier ein Fließkommazahl
  - „`itoa`“ etc. kein Standard! itoa-integer to ASCII
- **C++**
  - `std::stoi` (etc.) (C++11)
  - `std::to_string` (C++11)

Siehe https://www.cplusplus.com
oder https://en.cppreference.com/

C/C++-Wissen: to/from string

- **Besonderheit `std::to_string` (etc.) bei `float/double`**
  - Manche Standard-Funktionen funktionieren in Embedded-Bibliotheken nicht für `float/double` Typen (z. B. auch `printf`, `scanf`, ...)
  - Grund:
    - `float/double` in Embedded-Applikationen oft nicht genutzt
    - Unterstützung vergrößert Bibliothek signifikant
  - Workaround
    - Multiplizieren mit gewünschter Anzahl Nachkommastellen
    - Konvertieren zu Ganzzahl
    - Komma „manuell“ einfügen
  - Unterstützung in Startprojekt für Labor 3 „eingeschaltet“