Besondere Lernleistung

Das Übertragen eines Textes zwischen zwei Endgeräten, welcher in Bits umgewandelt und mit Hilfe der Manchester Codierung und der Java-Bibliothek RXTX über eine bestimmte Frequenz durch die serielle Schnittstelle geleitet und zum anderen Gerät geschickt wird. Danach wird der Bit Code entschlüsselt und als Text ausgegeben. Der serielle Datenaustausch wird durch das Implementieren einer GUI mit Java dargestellt.

Maximilian Schneider

Informatik

Städtisches Gymnasium Rheinbach

Fachlehrer: Rolf Faßbender, IF LK 1

Schuljahr 2020/2021

Inhaltsverzeichnis

Einleitung

Vor etwa 1,7 Millionen Jahren entwickelte sich die Hominiden (Urmenschen) weiter und waren in der Lage durch Gesten und Laute zu kommunizieren. Im Verlaufe der Zeit wurden die Hominiden zu Homo, welche dem heutigen Menschen ähnlicher waren. Sie verbesserten die Kommunikation mit Sprache durch mehr Laute und Gesten. Und so entwickelte sich die Sprache der Menschen weiter und weiter. Neue Kommunikationswege wurden erforscht und gefunden und somit auch schließlich die elektronische Kommunikation.

1971 wurde die erste E-Mail verschickt und ermöglichte der Welt einen modernen und schnellen Weg Informationen auszutauschen und zu übermitteln. Schließlich durch den ersten „Personal Computer“ (kurz PC) von IBM im Jahre 1981 war die Grundlage für die persönliche Kommunikation geschaffen.

Doch schon ein Jahrzehnt davor entstand die serielle Schnittstelle RS-232, welche in den 1960er Jahren als Schnittstelle für Datenübertragung zu Terminals und Eingabegeräten diente. Durch diese konnte mit Hilfe von Kabel Informationen zwischen verschiedenen Geräten ausgetauscht werden. Aber wie sieht dieser Datenaustausch aus und wie wurde er weiterentwickelt um anderen Varianten des Datenaustausches, wie der parallelen Schnittstelle, standzuhalten?

In dieser besonderen Lernleistung wird ein serieller Datenaustausch mit mehreren Endgeräten (Computern) verwirklicht und mit einer Graphischen Nutzerüberfläche (kurz GUI) veranschaulicht.

1.Allgemeines

Programmierung in Java

Im Rahmen dieser besonderen Lernleistung wird die Bitumwandlung und die GUI für den seriellen Datenaustausch ausschließlich in Java programmiert. Die Auswahl der Programmiersprache ist auf Java gefallen, da sie einfach strukturiert ist, objektorientiert arbeitet und plattformunabhängig ist.[1] [2]

Programmierung in IntelliJ IDEA

Da für dieses Projekt eine IDE (engl.: „integrated development environment“, dt.: „Integrierte Entwicklungsumgebung“) benötigt wird, ist die Auswahl auf die IDE IntelliJ IDEA gefallen. Meine Wahl fiel auf diese IDE, da ich diese bereits bei mehreren Projekten genutzt habe und IntelliJ IDEA eine Vielzahl an vorinstallierten Plugins besitzt. Fehler und Probleme werden live im Editor angezeigt und es werden Vorschläge zur Verbessrung bzw. zur Behebung der Fehler vorgeschlagen. Nichtsdestotrotz war es nicht leicht eine funktionierende GUI aufzustellen und diese benutzerfreundlich zu gestalten.

2. Die serielle Schnittstelle

Erklärung

Die serielle Schnittstelle ist im informellen Sprachgebrauch eine Bezeichnung für eine Schnittstelle zur Datenübertragung. Diese Datenübertragung verläuft normalerweise zwischen zwei Geräten. Die Übertragung erfolgt Bit für Bit nacheinander, was im Bereich der EDV (Elektronische Datenverarbeitung) als serieller Ablauf bezeichnet wird.

[1] Vgl. (3), „Vor- und Nachteile von JAVA“
[2] Vgl. (4), „Welche Vorteile bietet die Programmiersprache Java?“

Abbildung 1: Die serielle Schnittstelle an einem Endgerät [3]

Varianten

Die serielle Schnittstelle gibt es mit einem 9-poligen und einem 25-poligen Stecker. Der Hauptunterschied liegt darin, dass der 25-polige zwei Datenkanäle zur Verfügung stellt. Außerdem gibt es verschiedene Versionen der seriellen Schnittstelle, wie die RS-232 oder aber auch die RS-485.[4]

Entwicklung

Die in den 1960er Jahren entstandene serielle Schnittstelle, diente ursprünglich als Schnittstelle zu Terminals. Jedoch konnte die RS-232 auch den Datenaustausch mit Modems ermöglichen. Diese Modems setzten schließlich Nachrichten auf Telefonleitungen um. Eine große Aufmerksamkeit bekam die serielle Schnittstelle aber erst, als die Vernetzung von Terminals mit Rechnern möglich war. So konnten auch grafische Terminals sich mit schnellen Computern bzw. Rechnern verbinden. Die ursprüngliche Variante war ein 25-poliger Stecker, welcher jedoch in den heutzutage bekannten 9-poligen Stecker ausgewechselt wurde. Dies passierte auf Grund dessen, dass Leitungen zur synchronen Datenübertragung überflüssig waren und von UART nicht gebraucht bzw. nicht unterstützt wurde. Im Bereich der Computerspiele war früher das Kopf-an-Kopf-Spielen ein bekannter Anwendungsfall. Zwei Spieler saßen sich in einem Raum gegenüber und spielten an verschiedenen Computern. Dabei waren die Computer über serielle Schnittstellen verbunden.

Doch durch die Entwicklung vieler Geräte wurde es wichtiger immer höhere Datenmengen zu übertragen. Auf kurze Entfernung setzte sich die parallele Schnittstelle gegen die serielle Schnittstelle durch, da sie größere Mengen an Daten verarbeiten kann.

[3] Vgl. (1), „Serielle Schnittstelle“ (Bildverzeichnis)
[4] Vgl. (8) „RS-232“

Beide Schnittstellen wurden jedoch im Verlauf der Jahre weitgehend durch die USB-Schnittstelle ersetzt.[5] [6]

Probleme

Die serielle Schnittstelle verlor durch die neuere Variante, dem USB, immer mehr an Bedeutung und ist deshalb in den letzten Jahren kaum noch an Computern vorhanden. USB oder IEEE 1394, ein weiteres Kabel zur seriellen Datenübertragung, sind schneller und haben eine größere Reichweite mit einer kleineren Fehlerquote. Jedoch kann man sich mit einem Adapter aushelfen, welcher den USB-Anschluss zu einer seriellen Schnittstelle umwandelt. Einer der einzigen Vorteile gegenüber der USB-Schnittstelle ist, dass ein Standartanschluss der RS-232-Schnittstlle ein 15 Meter abgeschirmtes Kabel ohne Probleme nutzen kann. Ein USB-Kabel ist ca. auf 5 Meter beschränkt.[7] [8]

Benutzung heute

Obwohl serielle Schnittstellen schon seit 2005 nicht mehr wirklich in PCs verbaut werden, gibt es immer noch Anwendungsfälle, wo diese genutzt werden. Zum Beispiel gibt es Industriemessgeräte, verschiedene Barcodescanner, Überwachungskameras und Industrieanlagen, welche eine Verbindung durch ein Null-Modem-Kabel benötigen. [9] [10]

Funktion

Die serielle Schnittstelle wurde ursprünglich geschaffen, um Geräte wie Fernschreiber usw. über Modems miteinander kommunizieren zu lassen. Wenn die beiden Geräte miteinander verbunden werden sollen, kann man die Geräte nicht mit einem Standard RS-232 Kabel verbinden, denn die Geräte haben beide einen Steckeranschluss und das Standardkabel ist für ein Modem und ein Endgerät ausgelegt und hat somit einen Stecker und einen Buchsenanschluss. Außerdem würden die Sendesignale bei einem 1:1 Kabel des einen Gerätes auf die Sendeleitungen des anderen geschaltet werden und die Empfangsleitungen auf die Empfangsleitungen des anderen. Damit nun doch eine Kommunikation stattfinden kann müssen sogenannte Null-Modem Kabel genutzt werden. Bei einem Null-Modem-Kabel werden die Leitungen gekreuzt womit die

[5] Vgl. (5) „Was ist die serielle Schnittstelle?“
[6] Vgl. (6) „Definition des Null-Modem-Kabels“
[7] Vgl. (5) „Was ist die serielle Schnittstelle?“
[8] Vgl. (7) „Rs-232-Die serielle Schnittstelle“
[9] Vgl. (5) „Was ist die serielle Schnittstelle?“
[10] Vgl. (8) „RS-232“

Sendeleitungen auf die Empfangleitungen des anderen geschaltet werden und andersrum. [11]

Weitere Infos bei „Übertragungskabel“

3. Bit

Erklärung

Bit, aus dem englischen von dem Wort „Binary Digit“, hat in verwandten Fachgebieten unterschiedliche Bedeutungen. Er gilt als Maßeinheit für den Informationsgehalt und für die Datenmenge digital gespeicherter Daten und als Bezeichnung für eine Bezeichnung einer Binärzahl.

Binärzahlen werden in einem Binärsystem verwendet und stellen alle Zahlen durch eine Folge der Werte „0“ und „1“ dar. Ein Bit ist die kleinste Informationseinheit für einen Computer und eine zusammengehörige Folge von 8 Bits ist ein Byte. Mit einem Byte lassen sich 256 Zeichen darstellen. [12] [13]

Umwandlung von Text zu Bit in Java

Für die Umwandlung von Text in einen binären Code gibt es in Java bereits eine integrierte Bibliothek (java.io.UnsupportedEncodingException), welche die Umwandlung vereinfacht.

Der individuelle Text wird mit Hilfe der Methode „Text.getBytes("UTF-8")“ zu Bytes umgewandelt und diese werden in einem initialisierten Byte-Array gespeichert. Der Parameter „UTF-8“ ist dabei die Angabe in welchem Zeichenformat die Buchstaben kodiert werden. UTF-8 (Abkürzung für 8-Bit UCS Transformation Format) ist in den ersten 128 Zeichen gleich wie ASCII und eignet sich bei dem Verwenden von vielen westlichen Sprachen die Buchstaben besitzen, welche nur einen Byte Speicherbedarf benötigen. Deutsche Umlaute zum Beispiel brauchen jedoch schon zwei Byte und deshalb mehr Speicherbedarf.

---

[11] Vgl. (7) „Rs-232-Die serielle Schnittstelle“
[12] Vgl. (3) „Computernetzwerke - Von den Grundlagen zur Funktion und Anwendung“ (Buchquelle)
[13] Vgl. (12) „Grundlagen serieller Kommunikation“

Zur Visualisierung werden die Bytes mit Hilfe der Methode Integer.*toBinaryString*() zu einem String, eine endliche Folge aus einem definierten Zeichensatz, umgewandelt und dieser kann in der Konsole ausgeben werden.

```
public void TextToByte(String Text){
   byte[] infoBin = null;
   infoBin = Text.getBytes("UTF-8");
   for (byte b : infoBin) {
       String bin = Integer.toBinaryString(b);
       if (bin.length() < 7 )
           bin = "0" + bin;
       System.out.println("Buchstabe:" + (char) b + "-> "+ bin);
   }
}
```

Pseudocode der Umwandlung von Text zu Bit in Java

```
1000100 1100001 1110011 0100000 1101001 1110011

D       a       s               i       s

1110100 0100000 1100101 1101001 1101110 0100000

t               e       i       n

1000010 1100101 1101001 1110011 1110000 1101001

B       e       i       s       p       i
1100101 1101100

e       l
```

Beispiel eines Textes, welcher in Byte umgewandelt wurde

4. Datenübertragung – Seriell/Parallel

Bei der seriellen Kommunikation werden Daten über ein Kabel vom Sender zum Empfänger geleitet. Dabei gibt es mehrere Varianten wie die Bits transportiert werden. Im Kontrast zur seriellen Datenübertragung ist die parallele Datenübertragung eine alternative Übertragungsart.

Parallele Datenübertragung

Bei der parallelen Datenübertragung werden mehrere Bits gleichzeitig parallel übertragen. Das heißt es können pro Takt mehrere Bits verschickt und verarbeitet werden. Die Bits laufen dabei jeweils durch ein eigenes Kabel.

Serielle Datenübertragung

Bei der seriellen Datenübertragung werden Bits nacheinander, also seriell, übertragen. Das heißt, es kann pro Takt nur ein Bit verschickt und verarbeitet werden.

Vergleich

Bei der parallelen Datenübertragung werden mehrere parallel gelegte Kabel benötigt, da für jedes Datenbit eine eigene Leitung bereitstehen muss. 8 Bit bzw. 16 Bit parallel zu übertragen ist zwar noch rentabel und wird zum Beispiel beim Anschluss von Druckern eingesetzt. Bei einer noch größeren Bitübertragung jedoch würde die Übertragung ein sehr dickes Kabel erfordern, welches auf Grund von Kostengründen meistens nie produziert und verwendet wird. Bei der seriellen Datenübertragung kann man auf Übertragungsleitungen und Anschlusselektronik verzichten und ist deswegen meistens die kostengünstigere Variante.

Synchrone Übertragung

Bei der synchronen Datenübertragung wird die Übertragung einzelner Bits zwischen beiden Nutzern mit einem Taktsignal gehändelt. Dieses Taktsignal kann über eine eigene Schnittstellenleitung gesendet werden oder vom Empfänger aus dem Datensignal zurückgewonnen werden. Dann spricht man von „Taktrückgewinnung“. Durch die

Synchronisation mit dem Taktsignal können die Daten zur zeitlichen Taktänderung übertragen werden.

Asynchrone Übertragung

Die RS-232 Schnittstelle sendet die Daten asynchron, das heißt, es wird kein zusätzliches separates Taktsignal, das die einzelnen Bits markiert und den Empfänger mit dem Sender synchronisiert, mit übertragen.[14]

Datenübertragungsrate

Die Datenübertragungsrate zeigt wie viele Daten (Bits) in einem Zeitintervall (meistens Sekunde) übertragen werden. Die Einheit ist meistens Bit pro Sekunde (bit/s oder bps) oder bei einer höheren Datenübertragungsrate Kilobit, Megabit oder Gigabit pro Sekunde.

$$\text{Übertragungsrate } C = \frac{Datenmenge\ D}{Zeit\ t}$$

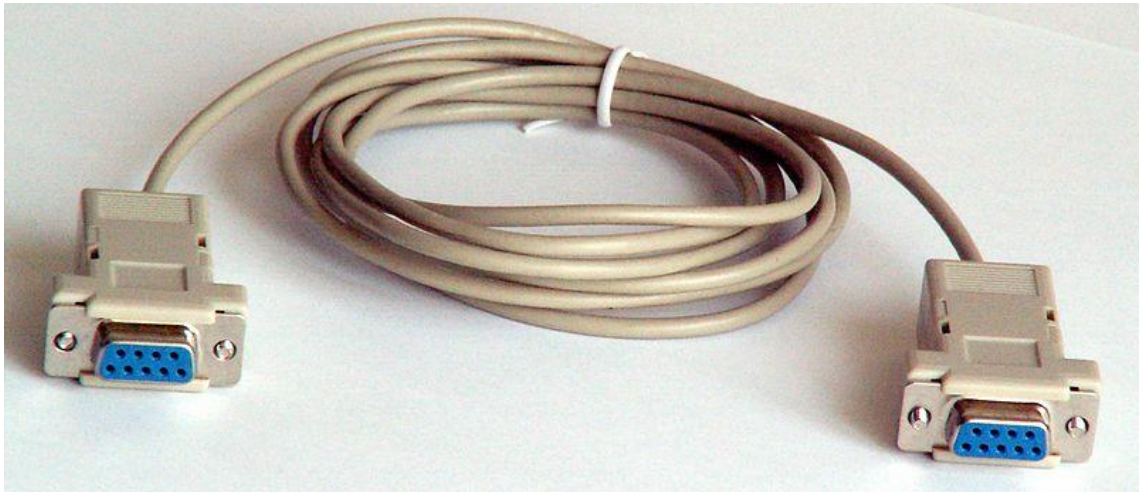
Aus dieser Formel lässt sich auch die Zeit t berechnen, welche die Datenmenge D bei einer Übertragungsrate C braucht.

$$Dauer\ t = \frac{Datenmenge\ D}{\text{Übertragungsrate } C}$$

Übertragungskabel und D-Sub Stecker/Buchse

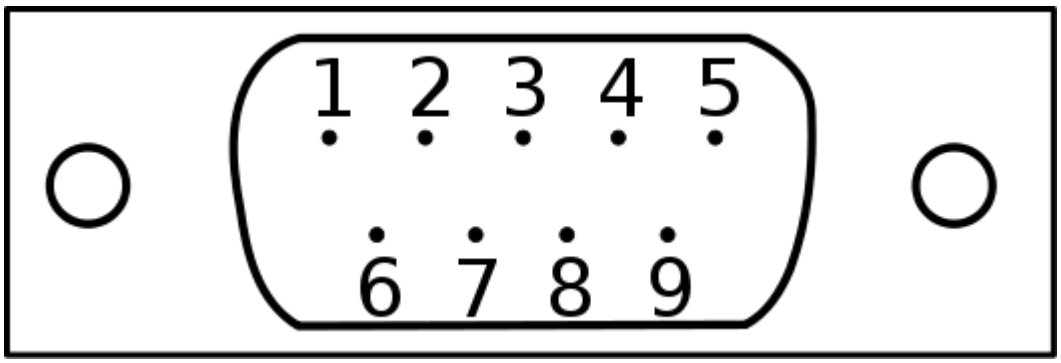
Das Verbindungskabel, welches zwei Endgeräte über die serielle Schnittstelle verbindet wird Nullmodemkabel genannt. Das normale serielle Verbindungskabel verbindet ein Endgerät mit einem Modem.

[14] Vgl. (7) „Rs-232-Die serielle Schnittstelle“



Ein typisches Nullmodemkabel[15]

Eine Buchse oder ein Stecker einer seriellen Schnittstelle hat 9 Pins. Diese werden nummeriert, damit eindeutig zu erkennen ist von welchem Pin gesprochen wird. In der unten angeführten Tabelle werden die Pin-Namen aufgeführt und wozu sie gebraucht werden.



Die Pinbelegung bei einem DTE (Computer oder Terminal)[16]

PIN	NAME	BEDEUTUNG
1	CD	Empfangsleitungssignal erkannt
2	RxD	Empfangsdaten
3	TxD	Sendedaten
4	DTR	Datenendgerät bereit
5	GND	Signalmasse
6	DSR	Übermittlungseinrichtung bereit
7	RTS	Sendeanforderung

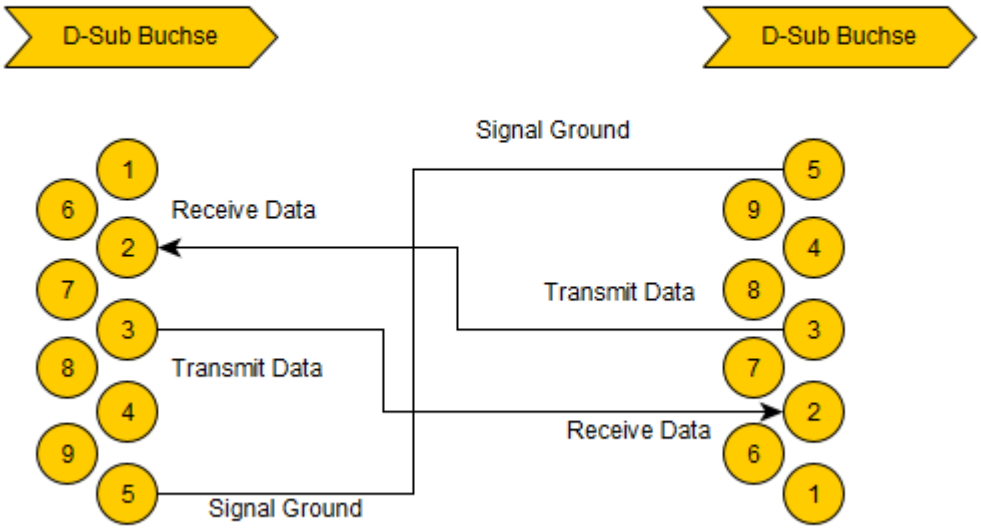
[15] Vgl. (2), "Nullmodemkabel" (Bildverzeichnis)
[16] Vgl. (3) „RS232 Buchse" (Bildverzeichnis)

8	CTS	Sendebereitschaft
9	RING	Rufzeichen

Serielle Verbindung

Eine serielle Datenverbindung zwischen zwei Geräten besteht grundsätzlich aus zwei Verbindungen für den Transfer von Bits und eine gemeinsame Verbindung für die Masse. Für den Transfer der Daten schickt jeder Computer die Bits immer von TxD (Transmit Data), dem Ausgangs-Pin zu RxD (Receive Data), dem Eingangs-Pin. Beide Verbindungen, Tx und Rx, können gleichzeitig genutzt werden. Die Verbindung der RxD und TxD Anschlüsse läuft über Kreuz, sodass der TxD Ausgang des ersten Systems mit dem RxD Eingang des zweiten Systems verbunden wird und umgekehrt. Die Anschlüsse müssen gekreuzt werden, da sonst die Computer die gleiche Sendeleitung und Empfangsleitung haben und so zwar Daten senden können, diese aber nie von dem anderen empfangen werden.

Die wichtigsten Pins sind also Pin 2 – RxD, Pin 3 – TxD und Pin 5 – GND.[17]



Verbindung zweier serieller Schnittstellen durch Nullmodemkabel

[17] Vgl. (14) „Uart Schnittstelle“

5.Uart

Uart (Universal Asynchronous Receiver Transmitter) ist ein im PCs und Mikrocontrollern standardisiertes elektronisches Bauelement bzw. Schaltung, welche zur Bewerkstelligung von seriellen Schnittstellen dient. Uart kann Daten über eine Datenleitung Senden und Empfangen und ist die Grundlage der seriellen Schnittstelle. UART wandelt die Daten aus Bytes in Bits um und sendet sie nacheinander über den Sendeteil (Tx) und empfängt über den Empfangsteil (Rx) und wandelt die ankommenden Bits in Bytes um.

Die Datenübertragung erfolgt asynchron, das heißt, es gibt kein Taktsignal zur Synchronisation, denn UART hat im Gegenzug zu anderen Transmittern keine zusätzliche Taktleitung. [18] [19]

Uart kann nur als Kommunikationsweg zwischen zwei Endgeräten bzw. Systemen benutzt werden. Aus diesem Grund gibt es bei den jeweiligen Systemen oft die Möglichkeit, dass dort mehrere Anschlüsse für mehrere Verbindungen vorhanden sind. Die Serielle Schnittstelle gibt es auch als Software-Emulator zum Installieren, da die Funktion und Vorgehensweise simpel sind.[20]

Duplex

Duplex bezeichnet das Konzept zur Aufteilung der Empfangs- und Senderichtung. Dabei kann zwischen zwei Optionen, dem Vollduplex- und dem Halbduplexverfahren, gewählt werden. Uart bietet die Möglichkeit, dass zwei Endgeräte mittels eines Null-Modem-Kabels über die RS-232 Schnittstelle im Vollduplex-Modus Daten austauschen können. Bei einem Full-duplex bzw. Vollduplex Verfahren läuft die Kommunikation über zwei getrennte, eigenständige Kanäle ab. Diese werden als Upstream und Downstream angegeben. Jeder Kommunikationsteilnehmer kann also Daten in beide Richtungen senden und empfangen. Da die Daten bei der seriellen Schnittstelle über zwei verschiedene Kabel empfangen und gesendet werden (RX und TX), ist ein Vollduplexbetreib möglich.

Angrenzend an das Vollduplexverfahren gibt es auch das Halbduplexverfahren, welches eine Kommunikation nur abwechselnd zwischen beiden Systemen über eine gemeinsame

[18] Vgl. (12) „Grundlagen serieller Kommunikation“
[19] Vgl. (14) „Uart Schnittstelle“
[20] Vgl. (7) „Rs-232-Die serielle Schnittstelle“

Leitung ermöglichen lässt. Dadurch können Kommunikationsteilnehmer Daten nur nacheinander senden und empfangen.[21]

Prinzip der Datenübertragung bei der seriellen Schnittstelle

Bei der RS-232-Schnittstelle erfolgt die Datenübertragung asynchron. Es gibt kein Taktsignal, weshalb der Empfänger also aus dem Datenfluss das Taktsignal zurückgewinnen muss. Die Synchronisation kommt durch die im Datenstrom enthaltenen Start und Stopp Bits zu Stande. Die asynchrone Übertragungsprozedur wird auch als Start-Stop-Verfahren bezeichnet, da vor dem Senden einer Dateneinheit ein Start-Signal geschickt wird, und nach dem Senden ein Stop-Signal. Zwischen diesen Signalen werden die Datenbits geschickt, welche eine Länge von 5-8 Datenbits haben.

Leitungscodierung

Um Bits über eine Leitung zu übertragen gibt es verschiedene Methoden, die dies bewerkstelligen. Jede Variante hat ihr Vor und Nachteile und sind unterschiedlich weit entwickelt.

Der Grundsatz jeder Leitungsübertragung ist der Wechsel eines Strompegels auf der Leitung. Uart kann den Signalpegel zwischen 3V und 15 V oder zwischen -3V und -15V setzten. Dieser Ausschlag kann von den Uarts interpretiert und genutzt werden. Ein hoher Pegel kann beispielsweise als „1“ oder „0“ wahrgenommen werden. Somit können Nullen und Einsen über die Leitung codiert werden. Die sendende Uart braucht eine Eingabe von Bits, die sie mithilfe des Signalpegels übertragen kann. Die Umwandlung der Bits zu einem richtigen Pegelwechsel wird von der Uart eigenständig gemacht. Die entgegengesetzte Uart nimmt den Pegelwechsel wahr und kann aus diesem Wechsel die Bits wiederherstellen.

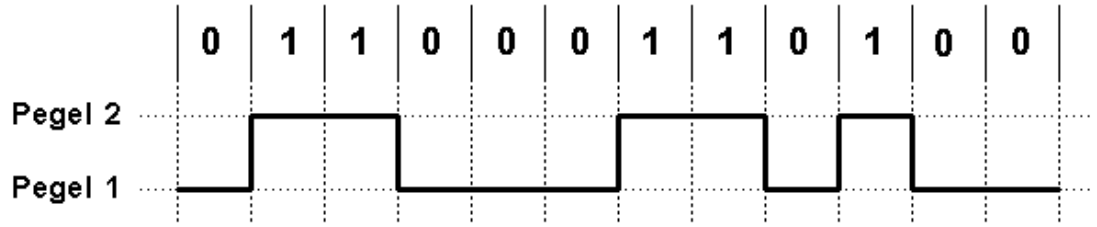
Es gibt verschiedene Codierungen, die bestimmen, wie Schnittstellen die Pegel verstehen und interpretieren.

NRZ-Codierung

Bei dem None-Return-To-Zero Verfahren (kurz NRZ-Verfahren) wird die Codierung durch die Erkennung des jeweiligen Signalpegels in einem Bitintervall ermöglicht. Eine „1“ wird durch einen hohen Pegel und eine „0“ wird durch einen negativen Pegel dargestellt. Der Signalwechsel findet immer an den Intervallgrenzen statt. Das Verfahren

[21] Vgl. (13) „Vollduplex“

ist Standard innerhalb digitaler Systeme, jedoch gibt es keine Taktrückgewinnung, da der Takt nicht aus der gegebenen Codierung zurückgewonnen werden kann. Des Weiteren ist die NRZ-Codierung genauso wie die RZ-Codierung nicht gleichanteilsfrei, was bedeutet, dass sich die Signalteile nicht ausgleichen und es nicht gleichviele hohe wie niedrige Pegel gibt.



NRZ-L-Code

Der NRZ-Code, Pegel 1 repräsentiert einen negativen Pegelzustand und Pegel 2 einen hohen Pegelzustand[22]

RZ-Codierung

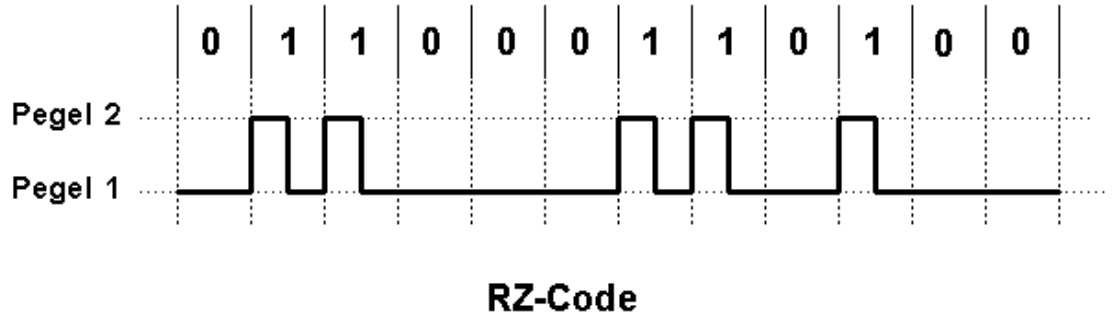
Bei dem Return-to-Zero-Verfahren (kurz RZ-Verfahren) wird die Übertragung ermöglicht, indem ein positiver Ausschlag des Stromes ähnlich wie beim NRZ als eine logische „1“ repräsentiert wird und wenn die Leitung einen negativen Ausschlag hat, wird dieser als eine logische „0“ wahrgenommen. Der Ausschlag des Stroms hat jedoch immer nur die Länge eines halben Taktintervalls und geht dann wieder in den Grundzustand über. Dadurch kann das Problem vermieden werden, dass bei einer langen gleichen Pegelfolge, die Uart bei dem Umwandeln fehlerhaft ist.

Nichtsdestotrotz ist eine Taktsynchronisation schwierig, denn bei langen Bitfolgen kann der Spannungsanteil sich verändern und die Umschaltflanke sich leicht verändern. Aber es wäre möglich ein Taktsignal durch den stetigen Wechsel in den Grundzustand zu kreieren. Aus diesem Grund wurde der Manchstercode entwickelt, der es ermöglicht, den Takt aus der Codierung, wiederzugewinnen.

Eine abgewandelte RZ-Codierung ist die Unipolare RZ-Codierung, wo das Low-Signal nicht codiert wird und es wird angenommen, dass wenn der Pegel im Grundzustand ist,

[22] Vgl. (6) „NRZ code“ (Bildverzeichnis)

dass dies als eine „0“ zu interpretieren ist. Es gibt nur den Grundzustand und einen hohen Pegelzustand. [23]



RZ-Code

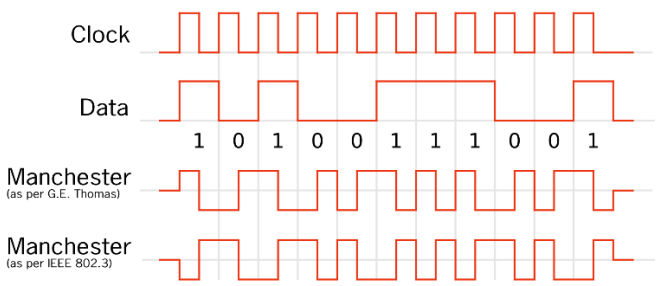
Die RZ-Codierung, Pegel 1 ist der Grundzustand und Pegel 2 ist der hohe postive Pegel[24]

Manchestercodierung

Der Manchester-Code ist ein Leitungscode, der aus einem Taktsignal und der Nutzdaten gebildet wird. Diese Verknüpfung wird auch Xor-Verbindung genannt. Die Informationen werden aus den Flanken, Übergänge zwischen den Signalzuständen, des Signals ausgelesen und treten in der Mitte einer Periode bzw. eines Intervalls auf. Nach Codedefintion von G.E. Thomas bedeutet eine fallende Flanke eine „1“ und eine steigende Flanke eine „0“. Eine weitere Codedefintion nach IEEE 802.3, der Ethernet-Technik, ist die Definition umgekehrt, das heißt, eine fallende Flanke bedeutet eine logische Null und eine steigende Flanke eine logische Eins. Dieses Verfahren wird auch die differenzielle Manchester Codierung genannt.

Die Verbindung der beiden Komponenten funktioniert, indem eine Xor-Verknüpfung aus Taktsignal und NRZ-codiertem Signal hergestellt wird. Das Taktsignal ändert jedes halbe Intervall den Pegel und dazu kommt nun das NRZ-Signal. Beim Taktende wird der Pegel immer auf den umgekehrten Pegelstand gestellt. Bei der Taktmitte folgt dann der Flankenwechsel und repräsentiert den Bitwert.[25] [26]

[23] Vgl. (22) „RZ-Code LinkFang“
[24] Vgl. (5) „RZ-Code“ (Bildverzeichnis)
[25] Vgl. (16) „Manchester-Codierung“
[26] Vgl. (17) „Leitungscodes zur Bitübertragung in Netzen“



Manchester Code in beiden Codevariationen[27]

Leitungscodierung der Seriellen Schnittstelle

Die Serielle Schnittstelle arbeitet mit dem NRZ-Leitungscode, da bei der Manchester Codierung die benötigte Bandbreite doppelt so hoch ist wie bei der NRZ-Codierung. Die Anzahl an einer höheren Bitzahl kommt zustande, da pro Nutzdatenbit 2 Codebits benötigt werden (1B2B-Codierung).

Ablauf einer Übertragung

Vor dem Start der Übertragung führt die Datenleitung im Ruhezustand den Pegel der logischen "1". Der Empfänger hört die Leitung ununterbrochen ab. Solange der Signalwert 1 gemessen wird, erfolgt keine Datenübertragung. Sobald die Übertragung beginnt, wird zunächst eine logische "0" gesendet, was den umgekehrten Pegel des Start-Bits darstellt. Der Empfänger erkennt die Änderung des Pegels und synchronisiert sich somit mit dem Sender. Als nächstes folgen die Datenbits der Nutzdaten. Die Datenbits werden abhängig von der Bitrate übertragen, wodurch der Empfänger weiß wann das nächste Bit übertragen wird. Aus diesem Grund ist es wichtig, dass beide Geräte die gleiche Bitrate haben, da sonst Sender und Empfänger nicht im gleichen Takt Bits übertragen und empfangen. Am Ende wird die Übertragung des Datensatzes mit einem Stopp-Bit, das durch eine logische "1" markiert ist, abgeschlossen. Die Dauer der Übertragungszeit hängt von der Bitrate ab. Das Stoppsignal kann nicht als Datenbit interpretiert werden, da beide Geräte vorher die Menge an Datenbits festgelegt haben, welche über die Leitung geschickt werden. Nach dem Empfang des Stop-Signals sind Sender und Empfänger wieder im gleichen Zustand wie zu Anfang und die nächsten Bits können übertragen werden.

[27] Vgl. (4) „Manchester encoding both conventions“ (Bildverzeichnis)

Das Stop-Signal dient dazu, dass der Empfänger Zeit hat die ankommenden Bits zu verarbeiten und sich auf die nächsten Bits vorzubereiten. Dadurch können potenzielle Fehler verhindert werden. Das nächste Start-Signals kann unmittelbar nach dem letzten Stop-Signal eingeleitet werden. Das Stop-Signal hat den Pegel “1”, da es mit dem Ruhezustand übereinstimmen soll. Wenn das Stop-Signal ausreichend lang ist, so kann der Empfänger das empfangene Zeichen ordnungsgemäß weiterleiten und seine Puffer, Zähler usw. für die Aufnahme des nächsten Zeichens vorbereiten.

Die 1 wird als Ruhezustand gewählt, da so klarer ist, wann der Pegel sich wechselt und weniger Fehler auftreten. Wenn der Ruhezustand bei einer 0 wäre, dann könnten kleine nicht beabsichtigte Stromstöße das Startbit auslösen, was zu Fehlern führen könnte.

Der RS-232-Schnittstelle hat keine voreingestellten Standardparameter. Alle Parameter wie Datenrate, Zeichenkodierung, Datenrahmen, Datenkompression, Fehlererkennung und Übertragungsprotokoll müssen vor der Datenübertragung auf beiden Geräten gleich eingestellt werden. Die erreichbare maximale Datenrate hängt von mehreren Faktoren ab. Im Wesentlichen sind die Schnelligkeit des Schnittstellenbausteins und die Länge und Qualität des Übertagungskabel die größten Einflussfaktoren. [28] [29]

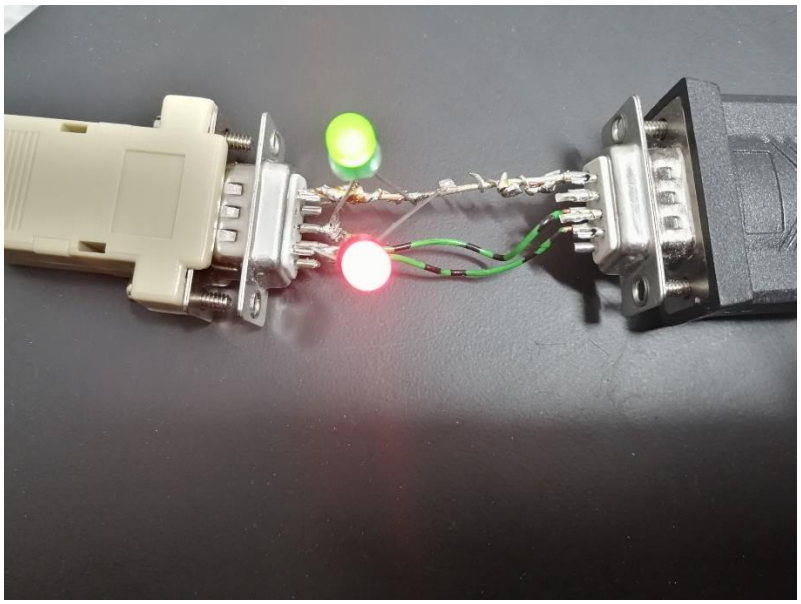
Nutzung von LEDs

Um die Übertragung zwischen zwei Geräten anschaulicher zu machen, werden LEDs genutzt, die die Übertragung zwischen den Endgeräten demonstrieren soll.

Dafür wird das Null-Modem-Kabel unterbrochen und durch eine angrenzende Buchse mit drei Kabel weitergeführt. Die drei Kabel sind die jeweiligen Rx bzw. Tx Leitungen der Geräte und die gemeinsame Masse-Leitung. Die LEDs sind mit der jeweiligen Anode an der Signal-Leitung und mit der Kathode an der Masse-Leitung verbunden.

Überträgt nun einer der beiden Computer Daten, wird durch den Pegelwechsel die zugehörige LED aufblinken und man könnte bei einer sehr niedrigen Baudrate, den Pegelwechsel ablesen. Jedoch ist die niedrigste Baudrate 100 und somit der Flankenwechsel unmöglich zu erkennen.

[28] Vgl. (18) „Asynchrone Übertragungsprozeduren“
[29] Vgl. (7) „Rs-232-Die serielle Schnittstelle“



Leitungen zwischen zwei Null-Modem-Kabel mit LEDs, die leuchten, sobald Daten ausgetauscht werden

6. Java-Bibliothek RXTX

Erklärung

Die Java-Bibliothek RXTX ist eine externe Bibliothek und ist nicht im normalen Package von Java enthalten. Aus diesem Grunde muss man sich die Bibliothek im Internet (zum Beispiel auf http://fizzed.com/oss/rxtx-for-java) herunterladen und dann zu dem aktuell genutzten Java-Package hinzufügen.

RXTX ist eine Bibliothek, welche eine serielle oder auch parallele Kommunikation bereitstellt. Die Bibliothek ist eine Erweiterung zum Java Development Toolkit, welche die serielle oder parallele Kommunikation nicht ohne zusätzliche Bibliothek ermöglicht.[30] [31]

Wichtige RXTX und Java Befehle

(Die genannten Befehle sind gekürzt und nicht identisch mit den Befehlen im Quellcode des Projekts, sondern dienen nur zur Veranschaulichung.)

[30] Vgl.(9) „Two way communication with serial port“
[31] Vgl. (10) „ RXTX for Java“

RXTX Bibliothek Benutzen:

Um die Bibliothek zu benutzen muss diese erstmal importiert werden. Dies passiert mit dem Befehl:

```
import gnu.io.*;
```

Com-Ports identifizieren:

Um herauszufinden welcher Com-Anschluss ein serieller ist wird mit Hilfe der folgenden Methode alle Com Ports getestet und serielle Ports zurückgeben.

```
public void CheckSeriellPort(){

    CommPortIdentifier serialPortId;
    //static CommPortIdentifier sSerialPortId;
    Enumeration enumComm;
    //SerialPort serialPort;

    enumComm = CommPortIdentifier.getPortIdentifiers();
    while (enumComm.hasMoreElements()) {
        serialPortId = (CommPortIdentifier) enumComm.nextElement();
        if(serialPortId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
            System.out.println(serialPortId.getName());
            SeriellPortName = serialPortId.getName();
        }
    }

}
```

Einen Port öffnen:

```
serialPortId.open("Project Name", Delay Time);
```

Mit diesem Befehl öffnet man einen Port. Dabei kann ein String, z.B der Project Name und eine Verzögerung angegeben werden.

```
outputStream = serialPort.getOutputStream();
```

Danach ist wichtig, dass der Output Stream, welcher die Daten versendet, auf den aktuellen Port gesetzt wird. Dadurch wird vermieden, dass der Output Stream von einem falschen und inaktiven Port Daten empfängt. Das gleiche gilt für den Input Stream:

```
inputStream = serialPort.getInputStream();
```

```
serialPort.addEventListener(new serialPortEventListener());
```

Zusätzlich muss nun auf den neuen Port ein EventListener initialisiert werden. Dieser erkennt sobald Daten über den Port geschickt und empfangen werden und kann darauf reagieren.

```
serialPort.setSerialPortParams(baudrate, dataBits, stopBits, parity);
```

Mit diesem Befehl werden die Parameter des Ports festgelegt.

Senden:

```
outputStream.write( StringName.getBytes() );
```

Das Senden wird beim Output Stream verwirklicht. Dieser bekommt Bytes in der Methode write() übergeben, welcher er dann versendet. Um aus einem String Bytes zu machen, braucht man die Methode getBytes().

Port schließen:

```
serialPort.close();
```

Sofern der Port geöffnet ist, wird dieser geschlossen und keine Daten können mehr über den Port empfangen und verschickt werden.

Daten erhalten:

Um Daten, die empfangen werden, zu erkennen, wird ein SerialPortEventListener benötigt. Dieser kann, sobald ein bestimmtes Ereignis auftritt, reagieren.

```
SerialPortEvent.DATA_AVAILABLE
```

Wenn der EventListener dieses Ereignis erkennt, heißt das, dass Daten von einem anderen Computer verschickt worden sind und nun über den Port eintreffen. Mit Hilfe der nächsten Befehle werden die Bytes empfangen und konvertiert:

```
while( inputStream.available() > 0 )
{
    int b = inputStream.read();
    input.add(b);

}
```

In dieser While-Schleife werden die ankommenden Bytes in einer Integer-ArrayList (input) gespeichert. Die bytes werden über den InputStream empfangen und können dort mit dem Befehl read() abgelesen werden.

```
String s = "";
for(int i=0; i<input.size(); i++)
{
   s += (char)( input.get(i).byteValue() );
}
```

In dieser for-Schleife werden die Bytes einzeln an einen String aneinander gehangen. Um die Bytes umzuwandeln wird die Methode byteValue() genutzt. Im Anschluss erhält man einen String der alle angekommenen und umgewandelten Bytes enthält. Dieser String ist die Nachricht, welcher der Sender geschickt hat.

Alternative zu RXTX

Die Bibliothek RXTX ist nicht die einzige Java-Bibliothek, welche den Zugang zu den seriellen Ports ermöglicht. JSerielCom zum Beispiel ist eine gute Alternative zu RXTX und wirbt vor allem mit besserer Benutzerfreundlichkeit, verbesserter Untersuchung für Zeitüberschreitungen und der Möglichkeit mehrere Ports gleichzeitig zu öffnen. [32]

Eigenschaften von Serial Port in Java

```
private Integer baudrate = 9600;
```

Die Baudrate gibt an wie oft ein Signal den Zustand ändert pro Sekunde. Der Standardwert liegt bei 9600 pro Sekunde. Die Datenübertragungsrate wiedrum gibt an wie viele Bits pro Sekunde übertragen werden. Bei einer Baudrate von 9600 braucht dann jedes Bit 104.2 µs zur Übertragung.

Die mögliche Baudrate hängt von der Kabellänge ab, denn mit steigender Kabellänge steigt auch die Fehleranfälligkeit. [33]

Baudrate (Signalwechsel pro Sekunde)	Kabellänge (in m)
2400	900
4800	300
9600	152
19200	15
57600	5
115200	<2

[32] Vgl. (11) „What is jSerialComm?“
[33] Vgl. (19) „Was ist der Unterschied zwischen Bitrate und Baudrate?“

```
Integer dataBits = SerialPort.DATABITS_8;
```

Mit diesem Befehl legt man die Standardlänge der Datenbits pro Byte fest. Im Normalfall liegt dieser zwischen 5 und 8. Bei diesem Projekt wird die Länge auf 8 Datenbits festgelegt, denn die Buchstaben bei UTF-8 brauchen 8 Bits.

```
private Integer stopBits = SerialPort.STOPBITS_1;
```

Mit diesem Befehl legt man die Standardanzahl von Stoppbits pro Byte fest.

Stoppbits, die am Ende jedes Zeichens gesendet werden, ermöglichen es der empfangenden Signalhardware, das Ende eines Zeichens zu erkennen und sich mit dem Zeichenstrom neu zu synchronisieren. Elektronische Geräte verwenden in der Regel ein Stoppbit. Bei langsamen elektromechanischen Fernschreibern werden eineinhalb oder zwei Stoppbits benötigt. [34]

```
private Integer parity = SerialPort.PARITY_NONE;
```

Parity (dt.: Parität) ist eine Möglichkeit zur Erkennung von Fehlern bei der Datenübertragung. Durch die Verwendung von Parität-Bits können fehlerhafte Datenbytes ausgemacht und erkannt werden. Die Methode funktioniert so, dass zu jedem Datenzeichen bzw. Datensatz ein zusätzliches Bit am Ende des Datenstranges angehängt wird, wodurch die Anzahl der Bits in jedem Zeichen entweder ungerade oder gerade sein müssen. Wenn also die Parität so eingestellt ist, dass die Anzahl immer gerade ist, werden nur Daten, die auch wirklich eine gerade Anzahl an Bits haben, akzeptiert. Andernfalls sind die Daten beschädigt worden, da ein Bit fehlt und werden somit nicht akzeptiert.

[34] Vgl. (20) „Stop-Bits“

Das Paritätsbit in jedem Zeichen kann auf einen der folgenden Werte eingestellt werden:

- **None:** None heißt, das kein zusätzliches Bit angefügt wird, also wird auf diese Art von Fehlererkennung verzichtet
- **Odd**: Odd heißt, dass die Anzahl an Bits ungerade sein muss, ansonsten werden die Bytes nicht akzeptiert und sind fehlerhaft
- **Even**: Even, heißt, dass die Anzahl an Bits gerade sein muss, ansonsten werden die Bytes nicht akzeptiert und sind fehlerhaft
- **Mark:** Mark heißt, dass der Paritäts-Bit immer eine logische 1 sein muss, ansonsten werden die Bytes nicht akzeptiert und sind fehlerhaft
- **Space:** Mark heißt, dass der Paritäts-Bit immer eine logische 0 sein muss, ansonsten werden die Bytes nicht akzeptiert und sind fehlerhaft

Die Mark und Space-Optionen sind eher unüblich, da sie keine zusätzlichen Informationen zur Fehlererkennung beitragen und deswegen oft weggelassen werden. Es gibt Sonderfälle, wo sie als neuntes Bit mitverschickt werden.

Die häufigste Paritätseinstellung ist jedoch “None”, da die Fehlererkennung auch durch ein Kommunikationsprotokoll erfolgen kann.

In diesem Projekt ist die Parity ebenfalls auf None, was heißt, dass keine extra Bits gesendet werden. Die Paritätsbits in diesem Fall nicht besonders wichtig, da nur geschrieben Texte übermittelt werden und es nicht schlimm ist, wenn Fehler auftreten. Zudem ist die Fehlerhäufigkeit gering, da nicht viele Daten übertragen werden müssen.[35]

[35] „Vgl. (21) „Parity“

7.GUI

Ziel dieser Lernleistung ist es, eine Serielle Datenübertragung mithilfe einer GUI zu vereinfachen und darzustellen. Dabei wird die Kommunikation zwischen zwei Endgeräten aber auch die zwischen mehr als nur zwei Geräten ermöglicht.

Erklärung der GUI:

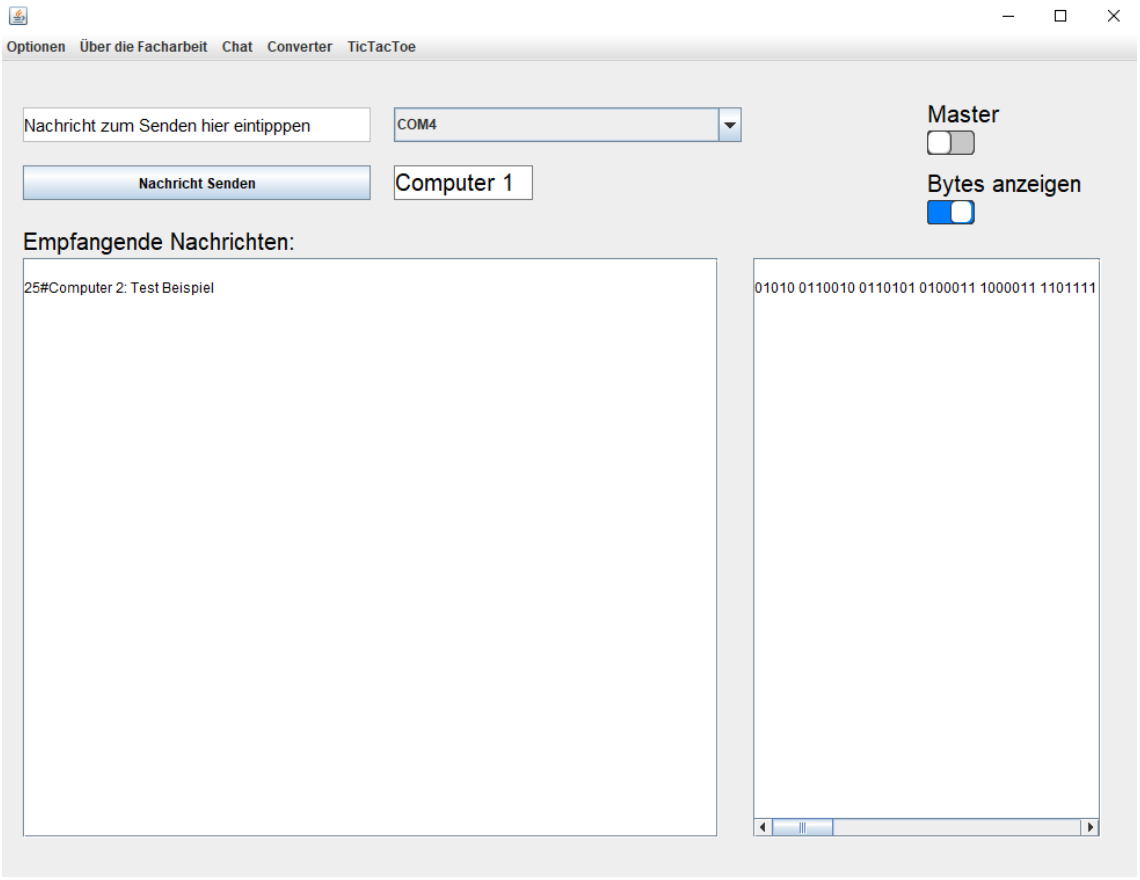
Im Folgenden werden die Funktionen und Bedienungselemente der GUI erklärt. Durch die Menüleiste oben an der GUI kann man zwischen den einzelnen Elementen wechseln.

Chat

Dies ist das Herzstück des Programms. Hier kann man mit dem anderen Computer kommunizieren. Die Nachricht, die gesendet werden soll, muss oben links in das Eingabefeld eingegeben werden. Rechts daneben kann man den Com-Port, an dem das Kabel angeschlossen ist, festlegen. Der Button unterhalb des Eingabefeldes sendet die zu sendende Nachricht. Das große Textfeld ist für die Visualisierung des Chats gedacht. Dort werden die Nachrichten, die man selbst schickt, aber auch die der anderen Computer angezeigt. Man kann seinen Namen bzw. die ID des Computers vorher angeben (Eingabefeld unter der Com-Auswahl). Wenn dies geschehen, wird vor jeder Nachricht angegeben, vom wem die Nachricht geschrieben wurde.

Der Masterswitch an der rechten Seite ermöglicht es, den nutzenden Computer als Master auszuwählen. Diese Funktion ist wichtig, falls mehr als nur zwei Computer Daten austauschen wollen. Ein Computer muss als Master ausgewählt werden, der die Daten, die ihm zugesendet werden, an alle weitergibt. Es kann nur einen Master geben.

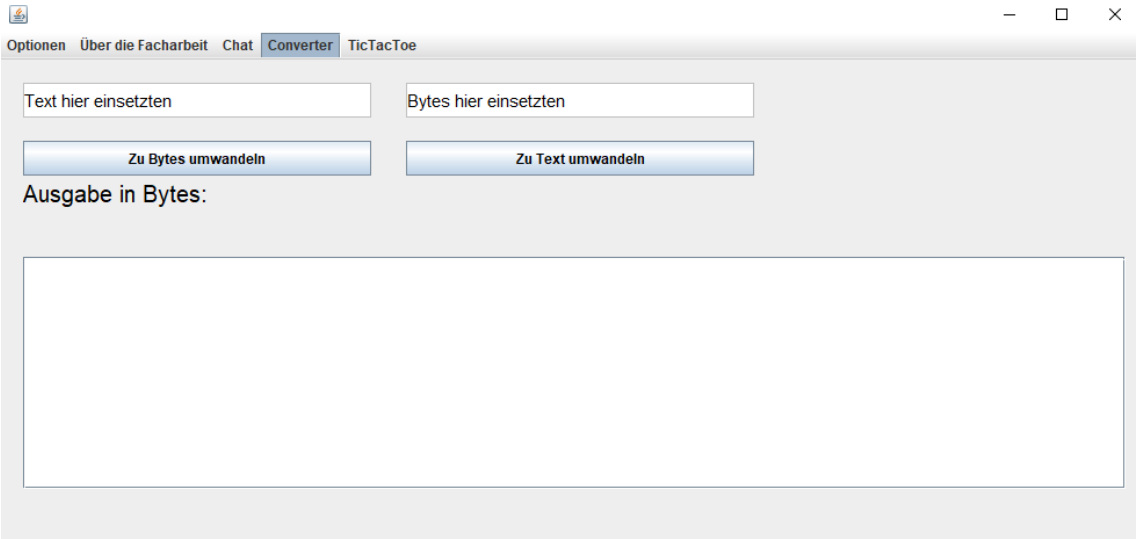
Es gibt einen weiteren Switch unter dem Masterswitch. Dieser Switch mit dem dazugehörigen Text „Bytes anzeigen“ ermöglicht es, die ankommenden Bytes anzuzeigen, bevor sie zum Text umgewandelt werden. Bei Betätigen des Switches wird ein weiteres Feld erzeugt, wo dann die Bytes in der selben Reihe wie der Text links daneben abgebildet werden.



Das Chat-Fenster in der GUI

Converter

Beim Converter kann zu Demonstrationszwecken Text in Bytes umgewandelt werden und umgekehrt. Dabei kann der zu übersetzende Text oben im linken Eingabefeld eingegeben werden und durch den Button darunter wird der Text schließlich in Bytes im Ausgabefeld unten ausgegeben. Im rechten Eingabefeld kann man Bytes eingeben und es kommt der dazugehörige Text raus.



Das Converter-Fenster in der GUI

Optionen

Hier können zusätzliche Optionen für die Datenübertragung geändert werden. Zum Beispiel kann man die Baudrate, die Parität und die Anzahl an Stoppbits ändern.

Das Optionen-Fenster in der GUI

TicTacToe

Als kleines Addon gibt es das Spiel TicTacToe, das über beide Computer gespielt werden kann. Man kann durch Drücken des Player-Buttons ändern, wer welcher Spieler ist und durch den Start-Button kann man das Spiel starten. Durch Drücken eines der neun Buttons kann man sein Zeichen an die gewählte Position setzen. Die nötigen Daten werden dann durch die Datenübertragung versendet.

Das TicTacToe-Fenster in der GUI

8. Verteiler bauen

Damit mehr als nur zwei Computer miteinander kommunizieren können, braucht man einen Verteiler, um die Signale von einem Computer auf die anderen zu leiten, denn ein einfaches Null-Modem Kabel reicht nicht aus. Die Signale eines Endgerätes müssen über ein Kabel geleitet werden, dass schließlich in allen anderen Endgeräten angeschlossen ist. Da ein normales Null-Modem-Kabel dies nicht bewerkstelligen kann, muss ein separater Verteiler gebaut werden. Dort kommt das Signal an einen Punkt an und wird dann durch den Verteiler an mehrere verschiedenen Kabel weitergeleitet. Das heißt, dass das Signal nicht nur in eine Richtung geschickt wird, sondern in mehrere Richtungen aufgeteilt wird. Um den Verteiler übersichtlich zu halten, werden nur drei weitere mögliche Richtungen eingebaut, an denen Endgeräte verkabelt sind. Dadurch erhält man ein Netz, an dem vier Rechner miteinander kommunizieren können. Dabei kann jedes Endgerät empfangen und senden.

Verkabelung und Bau

Bei dem Null-Modem-Kabel sind nur drei Pins zur Übertragung wichtig. Der GND (Masse), Rx und Tx (zur Übertragung). Diese Pins müssen also beim Verteiler beachtet werden. Um die einzelnen Pins zu erreichen muss an dem Verteilern 9 polige DSub Buchsen angebracht werden, wo die Null-Modem Kabel angeschlossen werden können. Von diesen DSub Buchsen kann dann jeder einzelne Pin angesteuert werden.

Um nun eine Kommunikation zu ermöglichen müssen die jeweiligen Anschlüsse aller Kabel miteinander verbunden werden. Jedoch gibt es Herausforderungen bei der Verkabelung. Man kann nicht einfach alle Rx-Pins mit allen Rx-Pins verbinden und alle Tx-Pins mit allen Tx-Pins. Die jeweiligen Pins dürfen nicht mit den anderen übereinstimmenden Pins verbunden werden, denn sonst käme keine Kommunikation zu Stande, da das Signal der Sendeleitung zum jeweiligen Rx-Pin muss, also zur Empfangsleitung. Daher muss eine andere Regelung her. Würde man nun versuchen ausgehend von jedem Computer den Tx-Pin mit allen Rx-Pins der anderen Computer zu verbinden wäre dies nicht möglich, denn dann wären alle Pins miteinander verbunden. Aufgrund dessen muss ein Computer ausgewählt werden, der als Master fungiert und die Kommunikation möglich macht.

Im Grunde ist der Master ein passiver Verteiler, der Daten an alle Geräte sendet. Der Master hat seinen Tx-Pin (Sendeleitung) mit allen Rx-Pins der anderen Geräte verbunden und kann so an alle Rx-Leitungen Daten senden. Die Leitungen werden über den Verteiler miteinander verbunden. Die anderen Geräte müssen nun auch Daten senden können. Dies wird so umgesetzt, dass alle Geräte, die nicht der Master sind, ihre Tx-Leitung mit der Rx-Leitung des Masters verbinden. So können alle Geräte an den Master schicken und der Master kann an alle Geräte schicken. Diese Geräte nennt man auch Slaves.

Mit diesem Modell können bisher aber alle Slaves nicht untereinander kommunizieren, weshalb eine weitere Funktion entstehen muss. Diese wird im Programm umgesetzt. In der GUI gibt es die Funktion, den Computer als Master auszuwählen, dadurch erkennt das Programm wer der Master ist. Das Programm ist so konstruiert, dass wenn der Slave in seiner übermittelten Nachricht klarstellt, dass diese Nachricht an alle und nicht nur an den Master gehen soll, dann nimmt der Master diese Nachricht auf und verschickt sie an alle weiter. Der Master dient so als Verteiler und jeder bekommt die verschickte Nachricht.

Der Bau des Verteilers ist dadurch schon vorgegeben. Zu beachten ist, dass an den Verteiler die Daten über ein Nullmodemkabel verschickt werden. Das heißt, dass die Leitungen gekreuzt werden und die Daten nicht beim eigentlichen Tx-Pin der DSub-Buchse ankommen, sondern beim Rx-Pin. Also muss auf Seite des Masters der zweite Pin mit allen dritten Pins der Slaves verbunden werden. Umgekehrt müssen alle zweiten Pins der Slaves mit dem dritten Pin des Masters verbunden werden.

Aber wie werden die Kabel verbunden?

Dafür gibt es mehrere Möglichkeiten. Einerseits kann die Verkabelung als Stern-Formation umgesetzt werden, wo sich die Kabel in der Mitte treffen (Siehe Verteiler 1). Andererseits kann man die Verkabelung auch als Bus/Linien-Formation umsetzten, wo die Kabel von D-Sub-Stecker zu D-Sub-Stecker verlegt werden. Beide Varianten haben ihre Vor-und Nachteile. Geht es primär um hohe Datenraten, ist die Line/Bus-Formation sehr gut geeignet, wenn es jedoch primär um ein geringer Energieverbrauch geht, ist die Stern-Formation besser geeignet.

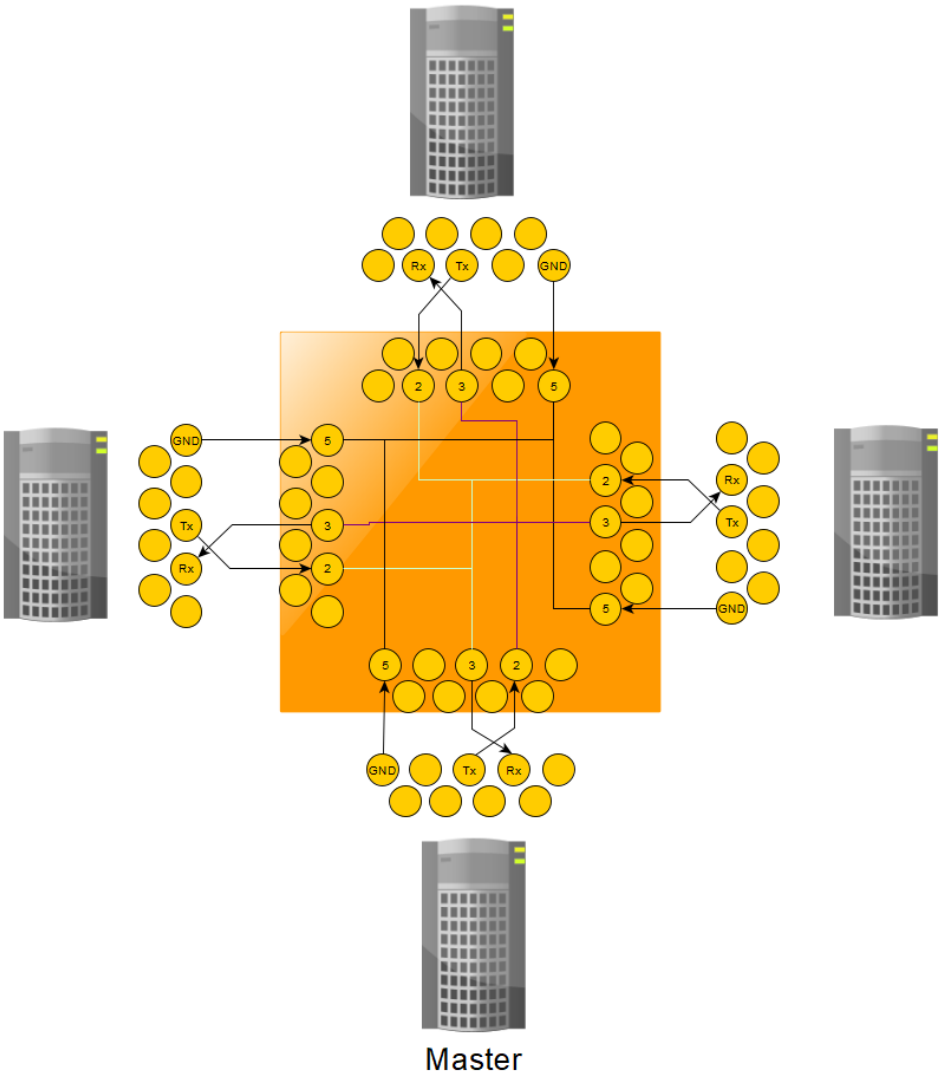
Des Weiteren will man Signalreflektionen vermeiden. Diese sind beim Linien-System unwahrscheinlicher und die Enden der Kabel können mit einem Widerstand gegen die Masse abgeleitet werden, wodurch Reflektionen verhindert werden. Aus diesem Grund ist eine Linien-Formation besser geeignet.

Wie bei Verteiler 2 zu erkennen, werden die Leitungen entlang der Buchsen geführt und gehen einmal im Kreis, bis sie bei einer Masseleitung abgeleitet werden. Im Bild sind die Grünen-Kabel, die Leitungen für die Datenübertragung vom Master zu den Slaves, die Blauen Kabel für die Masse und die roten Kabel für die Datenübertragung von den Slaves zum Master. An jede DSub-Buchse wird ein Nullmodem-Kabel angesteckt und das andere Ende in den PC geführt.

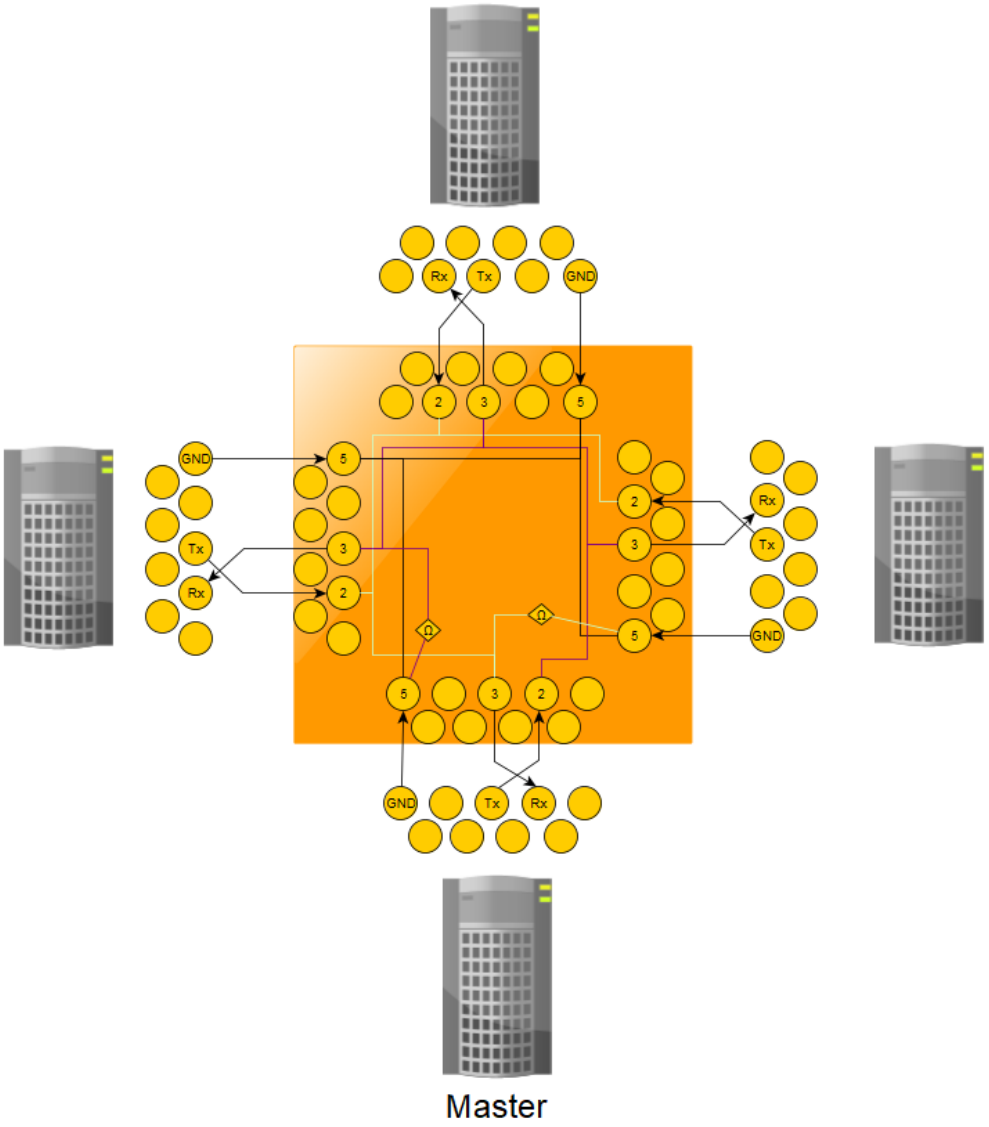
Ein Durchgang als Beispiel: der Master schickt ein Datensatz über ein Null-Modem-Kabel zum Verteiler. Die Daten werden beim Kabel gekreuzt und kommen am Verteiler beim Rx-Pin an (zweiter Pin des D-Sub-Steckers). Wären nun alle Rx-Pins der Stecker verbunden miteinander, würden die Daten dann bei den jeweiligen nächsten Null-Modem-Kabel, mit denen die Endgeräte verbunden sind, wieder gekreuzt werden und die Daten würden bei der Tx-Leitung ankommen. Die Endgeräte würden ewig auf Daten warten und nie welche erhalten. Um dies zu verhindern werden die Kabel auf dem

Verteiler erneut gekreuzt werden, damit die Daten auch wirklich auf der Rx-Leitung ankommen.

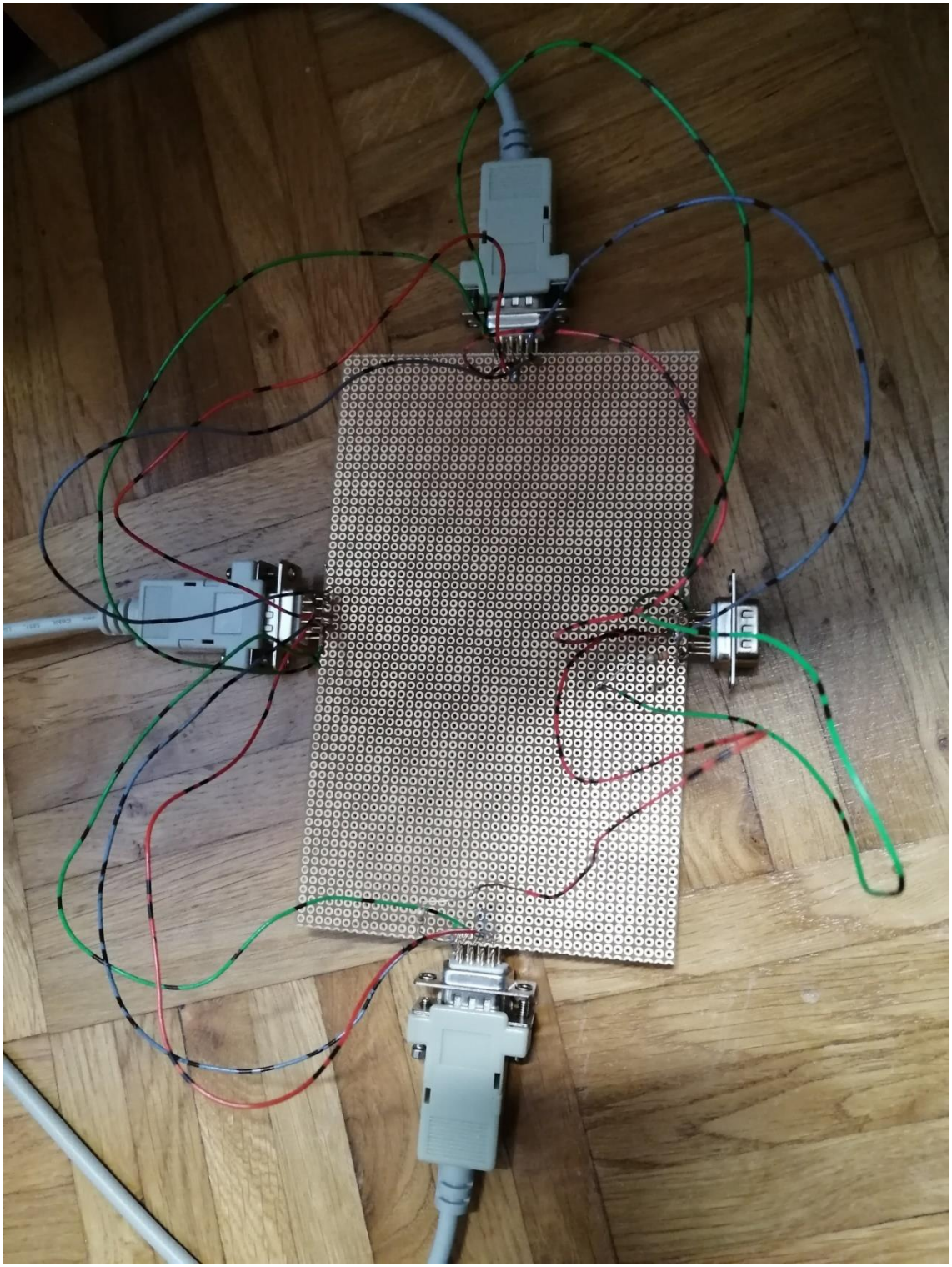
Der Master schickt also nun über seine Buchse am Verteiler vom zweiten Pin die Daten zum nächsten dritten Pin am Stecker eines Slaves. Dort werden die Daten geteilt und gehen einerseits über das Null-Modem-Kabel zum Slave, werden auf dem Weg gekreuzt und landen also auf dem zweiten Pin (Rx). Andererseits wandern die Daten auch zum nächsten Slave, wo sie ebenfalls geteilt werden und der gleiche Prozess wiederholt wird. Am letzten Slave werden die Daten, also der Strom durch einen Widerstand reduziert und gegen Masse abgeleitet, sodass sie verschwinden und es keine Reflektionen gibt. Auf diesem Weg erhalten alle Slaves eine Nachricht vom Master.



Verteiler 1 in Stern-Formation



Verteiler 2 in Linien-Formation



Verteiler von oben mit drei angeschlossenen Geräten, am unteren Ende ist der Master

RS-485

Die RS-232-Schnittstelle war nicht die letzte Schnittstelle die Entworfen wurde. Die Serielle Schnittstelle wurde immer weiterentwickelt und ein neueres Modell, die RS-485

Schnittstelle, wurde entworfen. Diese Schnittstelle hat einige Vorteile gegenüber der alten standardisierten RS-232 Schnittstelle.

Durch eine integrierte Mehrpunkttopologie können mehrere Empfänger und Sender leicht miteinander verbunden werden und die Übertragung ist konsistenter und stabiler. Außerdem ist die Kabellänge deutlich länger, denn die Schnittstelle kann bis zu 1,2 km fehlerfrei übertragen.

Durch Nutzung einer RS-485 Schnittstelle könnte man einen Kurzschluss durch Spannungsunterschiede vermeiden und die Übertragung mit mehreren Endgeräten wäre deutlich sicherer und einfacher.

Jedoch wurde bisher immer mit der RS-232 Schnittstelle gearbeitet und aus diesem Grund wird der Verteiler ohne die bessere Variante umgesetzt. [36]

9. Selbstreflexion und Selbstkritik

Mit dieser besonderen Lernleistung wurde beabsichtigt, die Datenübertragung mittels einer GUI darzustellen und eine serielle Datenübertragung durchzuführen. Beide Vorhaben sind aus eigner Perspektive sehr gut gelungen und die Datenübertragung wird ansprechend informativ illustriert.

Die Struktur und das Aussehen der GUI ist einfach gestaltet und wirft nicht viele Probleme auf. Die GUI ist funktional und so benutzerfreundlich wie möglich gestaltet. Eine nahezu perfekte GUI zu erstellen ist unmöglich, denn eine für alle Benutzer abgestimmte perfekte GUI zu erstellen ist nur schwer realisierbar. Nichtsdestotrotz ist die serielle Datenübertragung den Vorgaben entsprechend implementiert worden und funktionstüchtig. Die Übertragung ist so gut wie immer vollständig und beinhaltet nur selten Fehler. Die Byte Konvertierung ist in einer simplen GUI dargestellt. Des Weiteren gibt es änderbare Einstellungen für die Übertragung. Ein weiteres großes Ziel, war der Bau eines Verteilers. Dieser wurde vollständig konstruiert, entworfen, gelötet und zusammengebaut. Der Entwurf war aufwendig und die richtige Idee zu bekommen war schwer gewesen, denn es gab nicht viele Informationen und Texte über einen Verteiler im Internet oder in Büchern. Aber das stetige Auseinandersetzten mit dem Thema hat geholfen, dass nötige Verständnis zu sammeln. Der Verteiler und die dazugehörige

[36] Vgl. (15) „Unterschiede zwischen RS232 und RS485 - Anwendungsfälle und Technologie“

Programmierung sind wichtig für die Arbeit und haben mein Verständnis über die Serielle Schnittstelle tiefgründig verstärkt.

Die Serielle Schnittstelle nutzt das NRZ-Verfahren als Leitungscodierung und nicht den Manchestercode. Vor dem Erarbeiten des Themas war nicht klar, wie genau die Codierung der seriellen Schnittstelle aussieht und somit war es ungewiss ob der Manchestercode als Leitungscodierung genutzt wird. Vor Beginn der Arbeit war ich überzeugt, dass dies der Fall wäre und habe dementsprechend das Thema festgelegt. Im Nachhinein bin ich jedoch schlauer geworden und habe festgestellt, dass der Manchester-Code keine Rolle bei der seriellen Schnittstelle spielt. Dennoch wurde in dieser Lernleistung angesprochen und erklärt, denn er spielt eine große Rolle bei der Übertragung von Daten.

Literaturverzeichnis

Buchquellen:

[1]

Andrew S. Tanenbaum, „Computernetzwerke 4., überarbeitete Auflage“

ISBN 978-3-8273-7046-4

Pearson-Studium

[2]

James F. Kurose, Keith W. Ross „Computernetzwerke, Der Top-Down-Ansatz, 4., aktualisierte Auflage“

ISBN 978-3-8273-7330-4

Pearson-Studium

[3]

Rüdiger Schreiner, „Computernetzwerke - Von den Grundlagen zur Funktion und Anwendung, 3., überarbeitete Auflage“

ISBN 978-3-446-41922-3

Hanser-Verlag

[4]

Bernhard Hauser, „Fachwissen Netzwerktechnik – Modelle – Geräte – Protokolle“

ISBN 978-3-8085-5401-2

Europa Lehrmittel

Internetquellen:

[1]

Patrick Schnabel, „Serielle Schnittstelle (RS232/V.24/COM)“ (Zugriff am: 30.06.2020)

https://www.elektronik-kompendium.de/sites/com/0310301.htm

[2]

Wiesemann & Theis GmbH, „RS232-Schnittstelle - die Grundlagen“ (Zugriff am: 23.03.21)

https://www.wut.de/e-8wwww-16-apde-000.php

[3]

Uni-potsdam „Vor- und Nachteile von JAVA“ (Zugriff am: 30.06.2020)

http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/SeminarDidaktik/OOP/java_vor_nachteile.html

[4]

Christian Gesty, „Welche Vorteile bietet die Programmiersprache Java?“ (Zugriff am: 30.06.2020)

https://www.codingenieur.de/java/welche-vorteile-bietet-die-programmiersprache-java/

[5]

Eltima IBC, Was ist die serielle Schnittstelle?“ (Zugriff am: 30.06.2020)

https://www.virtual-serial-port.org/de/article/what-is-serial-port/

[6]

TechTarget, „Definition des Null-Modem-Kabels“ (Zugriff am: 30.06.2020)

https://www.computerweekly.com/de/definition/Null-Modem-Null-Modem-Kabel

[7]

InfoTip Service GmbH, „Rs-232-Die serielle Schnittstelle“ (Zugriff am: 30.06.2020)

https://kompendium.infotip.de/rs-232-die-serielle-schnittstelle.html

[8]

Wikimedia Foundation Inc., „RS-232“ (Zugriff am: 30.06.2020)

https://de.wikipedia.org/wiki/RS-232

[9]

RXTX org, „Two way communication with serial port“ (Zugriff am: 30.06.2020)

http://rxtx.qbang.org/wiki/index.php/Two_way_communcation_with_the_serial_port

[10]

Fizzed, „RXTX for Java“ (Zugriff am: 30.06.2020)

http://fizzed.com/oss/rxtx-for-java

[11]

Fazecast, Inc., „What is jSerialComm?“ (Zugriff am: 30.06.2020)

https://fazecast.github.io/jSerialComm/

[12]

Vogel Communications Group GmbH & Co. KG, „Grundlagen serieller Kommunikation“ (Zugriff am: 30.06.2020)

https://www.elektronikpraxis.vogel.de/raspberry-pi-grundlagen-serieller-kommunikation-a-824110/

[13]

BroadSoft Germany GmbH, „Vollduplex“ (Zugriff am: 30.06.2020)

https://www.placetel.de/ratgeber/vollduplex

[14]

Jirka Weissgärber, „Uart Schnittstelle“ (Zugriff am: 30.06.2020)

http://www.mathe-mit-methode.com/schlaufuchs_web/elektrotechnik/mikrocontroller_lernmaterial/microcontroller_allgemein/mikrocontroller_ext_hardware/mikrocontroller_uart.html

[15]

Eltima IBC, „Unterschiede zwischen RS232 und RS485 - Anwendungsfälle und Technologie“ (Zugriff am: 22.03.2021)

https://www.virtual-serial-port.org/de/article/what-is-serial-port/rs232-vs-rs485.html

[16]

DATACOM Buchverlag GmbH, „Manchester-Codierung” (Zugriff am:22.03.2021)

https://www.itwissen.info/Manchester-Codierung-Manchester-encoding.html

[17]

Detlef Mietke, „Leitungscodes zur Bitübertragung in Netzen“ (Zugriff am: 30.06.2020)

https://www.elektroniktutor.de/internet/codes.html

[18]
Uni Oldenburg, „Asynchrone Übertragungsprozeduren“ (Zugriff am: 30.06.2020)
https://einstein.informatik.uni-oldenburg.de/rechnernetze/asynchro1.htm

[19]
TechTarget, „Was ist der Unterschied zwischen Bitrate und Baudrate?“ (Zugriff am: 30.06.2020)

https://www.computerweekly.com/de/antwort/Was-ist-der-Unterschied-zwischen-Bitrate-und-Baudrate

[20]
Wikimedia Foundation Inc., „Stop-Bits“ (Zugriff am: 30.06.2020)

https://en.wikipedia.org/wiki/Serial_port#Stop_bits

[21]
Wikimedia Foundation Inc., „Parity“ (Zugriff am: 30.06.2020)

https://en.wikipedia.org/wiki/Serial_port

[22]

LinkFang.org, „RZ-Code LinkFang“ (Zugriff am: 30.06.2020)

https://de.linkfang.org/wiki/RZ-Code

Bildverzeichnis

[1]
Wollschaf, 10.Juli 2004: Serielle schnittstelle.jpg, Zugriff am: 06.07.2020
https://de.m.wikipedia.org/wiki/Datei:Serielle_schnittstelle.jpg
[2]
Ivan Pozdeev, 19. September 2007, „Nullmodemkabel“, Zugriff am: 12.08.2020
https://de.wikipedia.org/wiki/Datei:Null_modem_cable_1.jpg
[3]
En3rGy, 2. Mai 2011, „RS232 Buchse“, Zugriff am: 12.08.2020
https://de.wikipedia.org/wiki/Datei:RS232_Buchse_9pol_male.svg

[4]

Stefan Schmidt, 27 September 2006, „Manchester encoding both conventions“, Zugriff am: 22.03.21

https://commons.wikimedia.org/wiki/File:Manchester_encoding_both_conventions.svg

[5]

Ty von Sevelingen, 11 May 2004, „RZ-Code“, Zugriff am: 24.03.21

https://commons.wikimedia.org/wiki/File:RZ_code.png

[6]

Ty von Sevelingen, 11 May 2004 „NRZ code“, Zugriff am: 24.03.21

https://commons.wikimedia.org/wiki/File:NRZ_code.png

Anhang

A1: Umwandlungstabelle von Binär zu Dezimal (links das dazugehörige Zeichen)

Link: https://de.godaddy.com/blog/was-sind-eigentlich-ascii-code-und-ascii-tabelle/

Zeichen	**Binär**	**Dezimal**
nul	*0000 0000*	*0*
soh	*0000 0001*	*1*
stx	*0000 0010*	*2*
etx	*0000 0011*	*3*
eot	*0000 0100*	*4*
enq	*0000 0101*	*5*
ack	*0000 0110*	*6*
bel	*0000 0111*	*7*
bs	*0000 1000*	*8*
ht	*0000 1001*	*9*
lf	*0000 1010*	*10*
vf	*0000 1011*	*11*
ff	*0000 1100*	*12*
cr	*0000 1101*	*13*
so	*0000 1110*	*14*
si	*0000 1111*	*15*
dle	*0001 0000*	*16*
dc1	*0001 0001*	*17*
dc2	*0001 0010*	*18*
dc3	*0001 0011*	*19*
dc4	*0001 0100*	*20*

nak	*0001 0101*	*21*
syn	*0001 0110*	*22*
etb	*0001 0111*	*23*
can	*0001 1000*	*24*
em	*0001 1001*	*25*
sub	*0001 1010*	*26*
esc	*0001 1011*	*27*
fs	*0001 1100*	*28*
gs	*0001 1101*	*29*
rs	*0001 1110*	*30*
us	*0001 1111*	*31*
spc	*0010 0000*	*32*
!	0010 0001	33
„	0010 0010	34
#	0010 0011	35
$	0010 0100	36
%	0010 0101	37
&	0010 0110	38
,	0010 0111	39
(	0010 1000	40
)	0010 1001	41
*	0010 1010	42
+	0010 1011	43
,	0010 1100	44
–	0010 1101	45
.	0010 1110	46
/	0010 1111	47
0	0011 0000	48
1	0011 0001	49
2	0011 0010	50
3	0011 0011	51
4	0011 0100	52
5	0011 0101	53
6	0011 0110	54
7	0011 0111	55
8	0011 1000	56
9	0011 1001	57
:	0011 1010	58
;	0011 1011	59
<</p>	0011 1100	60
≤/p>	0011 1101	61
>	0011 1110	62
?	0011 1111	63
@	0100 0000	64
A	0100 0001	65
B	0100 0010	66
C	0100 0011	67
D	0100 0100	68
E	0100 0101	69

F	0100 0110	70
G	0100 0111	71
H	0100 1000	72
I	0100 1001	73
J	0100 1010	74
K	0100 1011	75
L	0100 1100	76
M	0100 1101	77
N	0100 1110	78
O	0100 1111	79
P	0101 0000	80
Q	0101 0001	81
R	0101 0010	82
S	0101 0011	83
T	0101 0100	84
U	0101 0101	85
V	0101 0110	86
W	0101 0111	87
X	0101 1000	88
Y	0101 1001	89
Z	0101 1010	90
[	0101 1011	91
\	0101 1100	92
]	0101 1101	93
^	0101 1110	94
_	0101 1111	95
`	0110 0000	96
a	0110 0001	97
b	0110 0010	98
c	0110 0011	99
d	0110 0100	100
e	0110 0101	101
f	0110 0110	102
g	0110 0111	103
h	0110 1000	104
i	0110 1001	105
j	0110 1010	106
k	0110 1011	107
l	0110 1100	108
m	0110 1101	109
n	0110 1110	110
o	0110 1111	111
p	0111 0000	112
q	0111 0001	113
r	0111 0010	114
s	0111 0011	115
t	0111 0100	116
u	0111 0101	117
v	0111 0110	118

w	0111 0111	119
x	0111 1000	120
y	0111 1001	121
z	0111 1010	122
{	0111 1011	123
:	0111 1100	124
}	0111 1101	125
~	0111 1110	126
del	0111 1111	127

Versicherung der selbständigen Erarbeitung

Ich versichere, dass ich die vorliegende Arbeit einschließlich evtl. beigefügter Zeichnungen, Kartenskizzen, Darstellungen u. ä. m. selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter genauer Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

Rheinbach, den 12.04.2020

(Ort) (Datum)

_____

(Unterschrift)