**Dokumentation**zum Projekt

***RFID***



|  |  |  |
| --- | --- | --- |
| Gruppe / Klasse | Mitarbeiter | Unterschrift |
| 5 / **4BHELS** | BIEHL S. |  |
| Übungs- / Abgabedatum | Mitarbeiter | Unterschrift |
| 17. Feb. 2015  8. Mai. 2015 | HOFSTÄTTER A. |  |
| Lehrer | Mitarbeiter | Unterschrift |
| Tillich / Gruber |  |  |
| Note | Mitarbeiter | Unterschrift |
|  |  |  |
| ***Projekt***  *RFID* | | |
| **Verwendete Geräte**   |  |  |  |  | | --- | --- | --- | --- | | Nr. | Gerät | Hersteller | Typ | | 1. | Netzgerät | EMG | 18135 | | 2. | Digital Multimeter | TE.Electronic | VA18B | | 3. | Oszilloskope | Tektronix | TDS 1001B |   **Verwendete Programme**   |  |  |  | | --- | --- | --- | | Nr. | Name | Version | | 1. | Altium Designer | 2015 | | | |

ÜBUNGS-/ABGABE-DATUM

Klasse /Gruppe

NOTE

LEHRER

# Inhaltsverzeichnis

[1 Inhaltsverzeichnis 2](#_Toc418842358)

[2 Aufgabenstellung 4](#_Toc418842359)

[2.1 Manchestercodierung 4](#_Toc418842360)

[3 Grundprinzip 5](#_Toc418842361)

[3.1 RFID 5](#_Toc418842362)

[3.2 Manchester-Codierung 5](#_Toc418842363)

[3.2.1 Allgemeines 5](#_Toc418842364)

[3.2.2 Vorteile 5](#_Toc418842365)

[3.2.3 Nachteile 6](#_Toc418842366)

[3.2.4 Weiteres 6](#_Toc418842367)

[4 Hardware 7](#_Toc418842368)

[4.1 Schaltung 7](#_Toc418842369)

[4.1.1 Reader (Sende-Empfänger) 7](#_Toc418842370)

[4.1.1.1 Funktionsweise 7](#_Toc418842371)

[4.1.2 Tag (Transponder) 7](#_Toc418842372)

[4.1.2.1 Funktionsweise 8](#_Toc418842373)

[4.2 Testlauf mit Funktionsgenerator 8](#_Toc418842374)

[4.2.1 Primärseitige Einspeisung 9](#_Toc418842375)

[4.2.1.1 Messpunkte 9](#_Toc418842376)

[4.2.1.1.1 Reader 9](#_Toc418842377)

[4.2.1.1.2 Tag 9](#_Toc418842378)

[4.2.1.2 Messergebnisse 10](#_Toc418842379)

[4.2.1.2.1 U1 & U2 (AM Trigger) 10](#_Toc418842380)

[4.2.1.2.2 U1 & U2 (Normaler Trigger) 10](#_Toc418842381)

[4.2.1.3 U3 11](#_Toc418842382)

[4.2.1.4 U4 (UA) 11](#_Toc418842383)

[4.2.1.5 Messpunkte 12](#_Toc418842384)

[4.2.1.5.1 Reader 12](#_Toc418842385)

[4.2.1.5.2 Tag 12](#_Toc418842386)

[4.2.1.6 Messergebnisse 13](#_Toc418842387)

[4.2.1.6.1 U1 & U2 (AM Trigger) 13](#_Toc418842388)

[4.2.1.6.2 U1 & U2 (Normaler Trigger) 13](#_Toc418842389)

[4.2.1.6.3 U3 & U4 14](#_Toc418842390)

[4.2.1.6.4 Eingangssignal & U4 (Ausgangssignal) 14](#_Toc418842391)

[5 Software 15](#_Toc418842392)

[5.1 Librarys 15](#_Toc418842393)

[5.1.1 LCD.h 15](#_Toc418842394)

[5.1.2 Matrixtastatur.h 15](#_Toc418842395)

[5.1.3 Manchester.h 15](#_Toc418842396)

[5.2 Programmlisting 15](#_Toc418842397)

[5.2.1 LCD.h 15](#_Toc418842398)

[5.2.2 Matrixtastatur.h 19](#_Toc418842399)

[5.2.3 Manchester.h 20](#_Toc418842400)

[5.3 Sende-Empfänger (Reader) 21](#_Toc418842401)

[5.3.1 C-File 21](#_Toc418842402)

[5.3.2 Header-File 25](#_Toc418842403)

[5.4 Tag 26](#_Toc418842404)

[5.4.1 C-File 26](#_Toc418842405)

[5.4.2 Header-File 28](#_Toc418842406)

[6 Abbildungsverzeichnis 30](#_Toc418842407)

# Aufgabenstellung

Es sollte ein RFID System zum drahtlosen Senden und Empfangen von Daten gebaut werden. Sender und Empfänger wurden jeweils unabhängig auf einem Steckbrett realisiert. Die Kopplung beider Komponenten erfolgte durch ein magnetisches Wechselfeld, welches in diesem Fall über zwei baugleiche selbstgewickelte Spulen erzeugt wurde.

Ziel des Projektes war es Befehle (z.B. Texte) an ein LCD-Display über eine Matrixtastatur zu übergeben. Die Datenübertragung sollte hierbei mit Hilfe einer AM-Modulation erfolgen, welche schließlich durch die Spulen übertragen wurden.

Auf jeder Seite (Primär/Sekundär) wurde ein AVR (ATmega32U4) platziert. Die Energieversorgung erfolgte nur von einer Seite (Primär), die Sekundärseite (inkl. Matrixtastatur und AVR) wurde drahtlos versorgt.

## Manchestercodierung

Um eine binäre Datenübertragung (mit beliebigen HIGH-LOW Verhältnissen) zu ermöglichen und über die gleiche Art auch die Sekundseite zu versorgen, darf das Datensignal nicht allzu lange auf logisch „LOW“ sein. Dies würde einen Einbruch der sekundärseitigen Spannung zur Folge haben. Da solch eine Art von Daten nicht garantiert werden konnte, mussten die Daten codiert werden. In diesem Projekt wurde die Manchestercodierung gewählt.

# Grundprinzip

## RFID

RFID (**R**adio-**F**requency **ID**entification)bezeichnet eine Technologie für Sender-Empfänger-Systeme zum automatischen und berührungslosen Identifizieren und Lokalisieren von Objekten und Lebewesen mit Radiowellen.

Ein RFID-System besteht aus einem Transponder (umgangssprachlich auch Funketikett genannt), der sich am oder im Gegenstand bzw. Lebewesen befindet und einen kennzeichnenden Code enthält, sowie einem Lesegerät zum Auslesen dieser Kennung.

RFID-Transponder können so klein wie ein Reiskorn sein und implantiert werden, etwa bei Menschen oder Haustieren. Darüber hinaus besteht die Möglichkeit, RFID-Transponder über ein spezielles Druckverfahren stabiler Schaltungen aus Polymeren herzustellen. Die Vorteile dieser Technik ergeben sich aus der Kombination der geringen Größe, der unauffälligen Auslesemöglichkeit (z. B. bei dem am 1. November 2010 neu eingeführten Personalausweis in Deutschland) und dem geringen Preis der Transponder (teilweise im Cent-Bereich). Diese neue Technik kann den heute noch weitverbreiteten Barcode ersetzen.

Die Kopplung geschieht durch vom Lesegerät erzeugte magnetische Wechselfelder geringer Reichweite oder durch hochfrequente Radiowellen. Damit werden nicht nur Daten übertragen, sondern auch der Transponder mit Energie versorgt. Nur wenn größere Reichweiten erzielt werden sollen und die Kosten der Transponder nicht sehr kritisch sind, werden aktive Transponder mit eigener Stromversorgung eingesetzt.

## Manchester-Codierung

Der Manchester-Code ist ein Leitungscode, der bei der Kodierung das Taktsignal erhält. Dabei moduliert eine Bitfolge binär die Phasenlage eines Taktsignals. Der Manchester-Code stellt damit eine Form der digitalen Phasenmodulation dar, welche auch als Phase Shift Keying bezeichnet wird.

Anders ausgedrückt tragen die Flanken des Signals, bezogen auf das Taktsignal, die Information.

### Allgemeines

Es gibt für den Manchester-Code zwei mögliche und gleichwertige Definitionen, wie auch in der nebenstehenden Abbildung dargestellt:

Eine fallende Flanke bedeutet in der Codedefinition nach G.E. Thomas eine logische Eins, eine steigende Flanke eine logische Null. Diese Definition wird auch als Biphase-L oder Manchester-II bezeichnet.

In der Codedefinition nach IEEE 802.3, wie sie bei 10-Mbit/s-Ethernet verwendet wird, bedeutet eine fallende Flanke eine logische Null und eine steigende Flanke eine logische Eins.

In jedem Fall gibt es mindestens eine Flanke pro Bit, aus der das Taktsignal abgeleitet werden kann. Der Manchester-Code ist selbstsynchronisierend und unabhängig vom Gleichspannungspegel. Um dem Empfänger mitzuteilen, wie im Signal eine logische Eins codiert ist, wird zu Beginn einer Datenübertragung ein Header (Präambel) versendet.

### Vorteile

Eine wesentliche Eigenschaft dieses Leitungscodes ist die Gleichanteilsfreiheit des resultierenden Signals. Dies bedeutet, dass der Gleichspannungsanteil genau null ist. Daher ist es möglich, die Signalfolge beispielsweise über Impulstransformatoren mit einer galvanischen Trennung zu übertragen.

Ein weiterer Vorteil ist, dass, wie oben beschrieben, aus dem Code selbst das Taktsignal abgeleitet werden kann. Ein zusätzlicher Taktgeber wird nicht benötigt.

### Nachteile

Ein Nachteil der Manchester-Codierung ist, dass bei der Datenübertragung die benötigte Bandbreite doppelt so hoch ist wie bei der einfachen Binärcodierung (z. B. Non Return to Zero, NRZ). Der Grund dafür liegt darin, dass für die Codierung eines Bits zwei Signale benötigt werden. Die Bitrate (im Fall eines zweiwertigen Signals) ist somit nur halb so groß wie die Baudrate.

### Weiteres

Neben dem Manchester-Code gibt es noch den differentiellen Manchester-Code. Bei diesem findet, im Gegensatz zur Manchester-Codierung, bei einem bestimmten Bit, meist ist es logisch Eins, ein Phasenwechsel statt. Bei logisch Null erfolgt kein Phasenwechsel. Dadurch geht die feste Zuordnung zwischen Richtung des Flankenwechsels und logischem Signalzustand verloren und es kann somit auch bei invertiertem Signal die Information richtig interpretiert werden.

# Hardware

Aufgrund der Die Hardware brauchte keine Dimensionierungen bzw. Berechnungen, weil bereits alle Werte vorgegeben waren. Die einzelnen Projekte unterschieden sich rein durch die Software.

## Schaltung

### Reader (Sende-Empfänger)

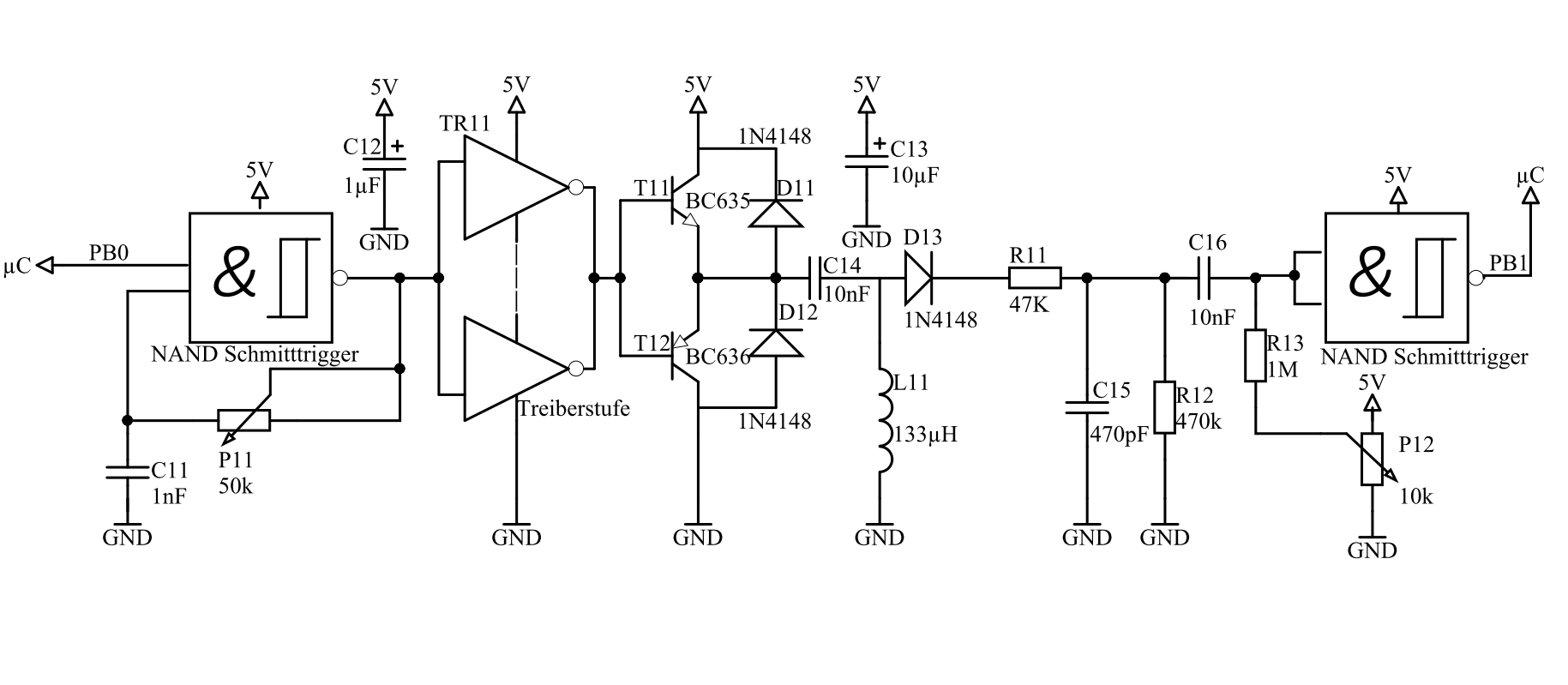


Abbildung 1. - Schaltung des Readers (Primär)

#### Funktionsweise

Das Lesegerät (Reader), das je nach Typ auch Daten schreiben kann, erzeugt ein hochfrequentes elektromagnetisches Wechselfeld, dem der RFID-Transponder (RFID-Tag) ausgesetzt wird. Die von ihm über die Antenne aufgenommene Hochfrequenzenergie dient während des Kommunikationsvorganges als Stromversorgung für seinen Chip. Der so aktivierte Mikrochip im RFID-Tag decodiert die vom Lesegerät gesendeten Befehle. Die Antwort codiert und moduliert das RFID-Tag in das eingestrahlte elektromagnetische Feld durch Feldschwächung im kontaktfreien Kurzschluss oder gegenphasige Reflexion des vom Lesegerät ausgesendeten Feldes.

### Tag (Transponder)

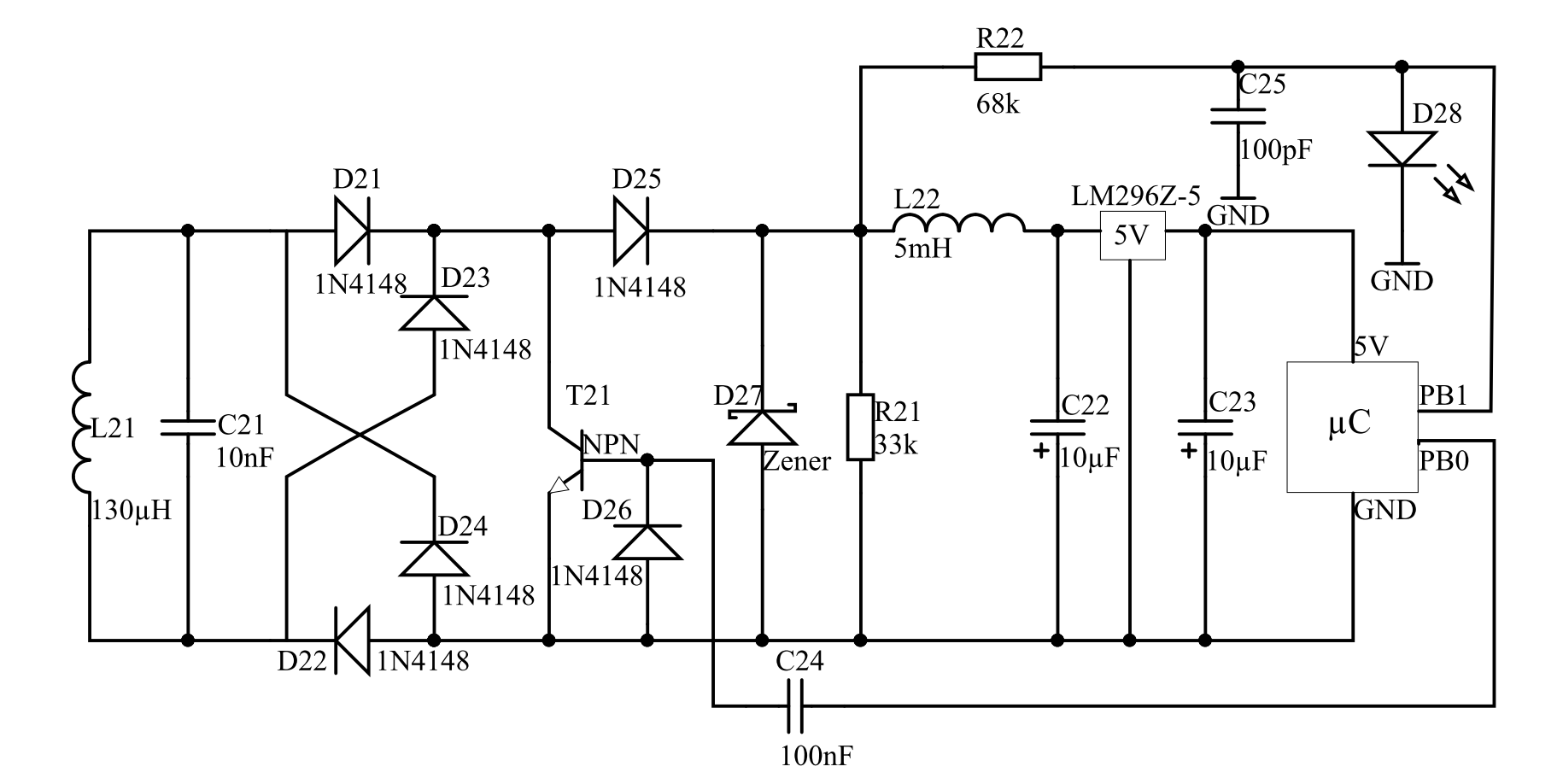


Abbildung 2. - Schaltung des Tags (Sekundär)

#### Funktionsweise

Der RFID-Tag arbeitet je nach Typ im Bereich der Langwelle, in diesem Fall zwischen (130-150)kHz. Die von ihm über die Antenne aufgenommene Hochfrequenzenergie dient während des Kommunikationsvorganges als Stromversorgung für seinen Chip. Der Tag erzeugt selbst also kein Feld, sondern beeinflusst das elektromagnetische Sendefeld des Readers.

## Testlauf mit Funktionsgenerator

Bevor die eigentliche Übertragung mit µControllern stattgefunden hat, wurden 2 Testläufe mit Hilfe von einem Funktionsgenerator durchgeführt. Das Signal wurde einmal primärseitig und einmal sekundärseitig eingespeist um die Funktion der Hardware zu testen. Es wurden jeweils verschiedene Signale gemessen um die Tauglichkeit zu prüfen.

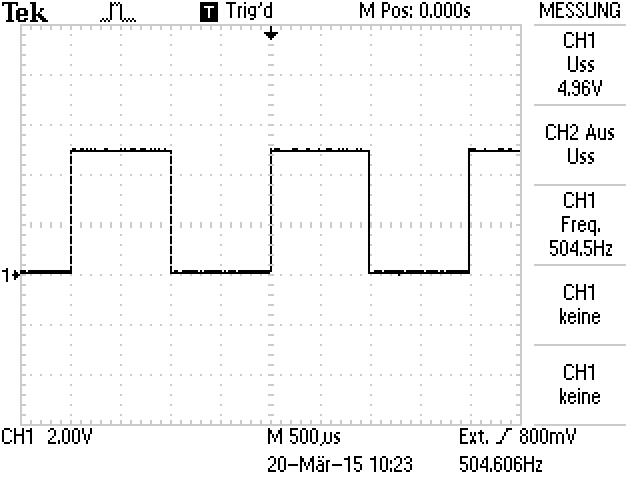


Abbildung 3. - Primär/Sekundär eingespeistes Signal des Funktionsgenerators

Das Signal entspricht dem des späteren verwendeten µController (5V, 500Hz, ton = toff).

### Primärseitige Einspeisung

Alle Signale wurden mit einem Oszilloskop und Tastköpfen gemessen.

#### Messpunkte

##### Reader

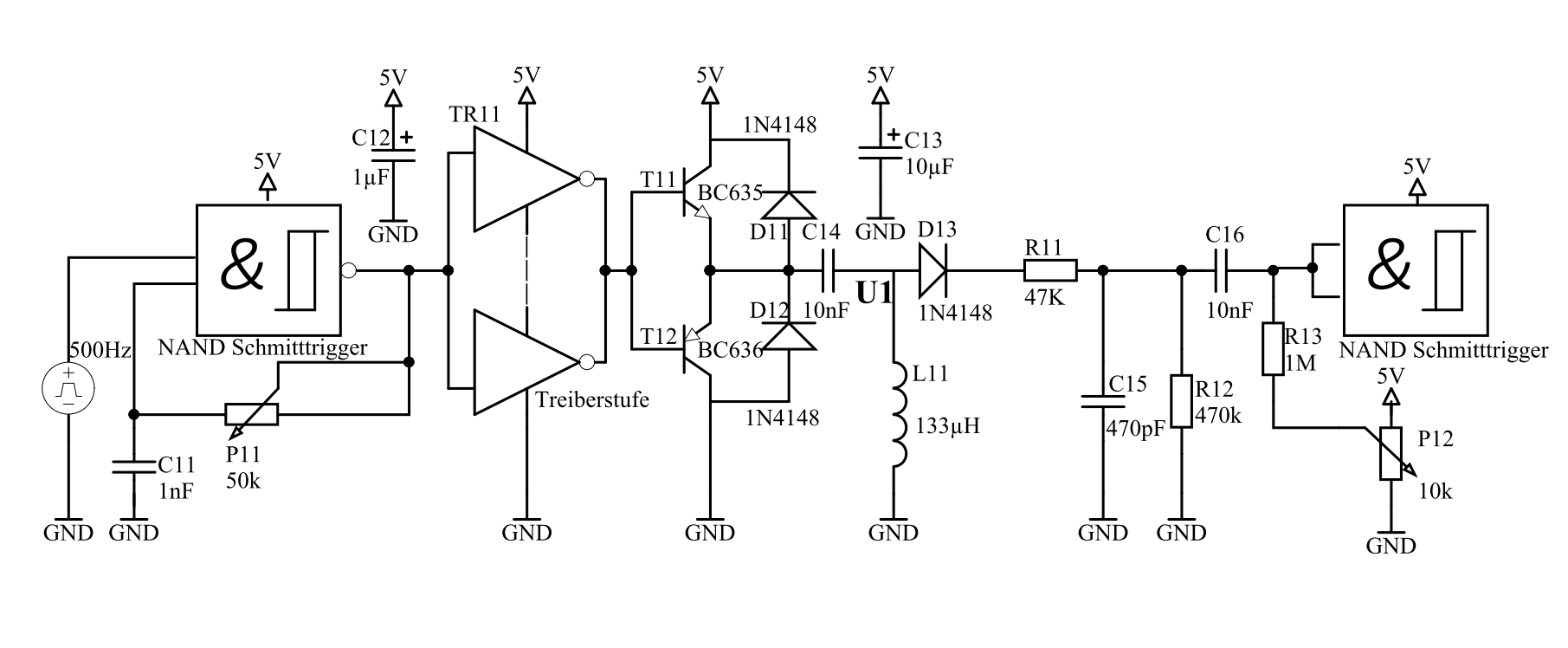


Abbildung 4. - Messpunkte Reader Primäreinspeisung

Am Eingang wurde der µController durch einen Funktionsgenerator ersetzt. Am Ende wurde ebenfalls entfernt, weil nur Sekundärseitig empfangen wird. Der Messpunkt U1 zeigt das übertragene Signal (es wurde zwischen Kondensator C14 und Spule L11 gemessen).

##### Tag

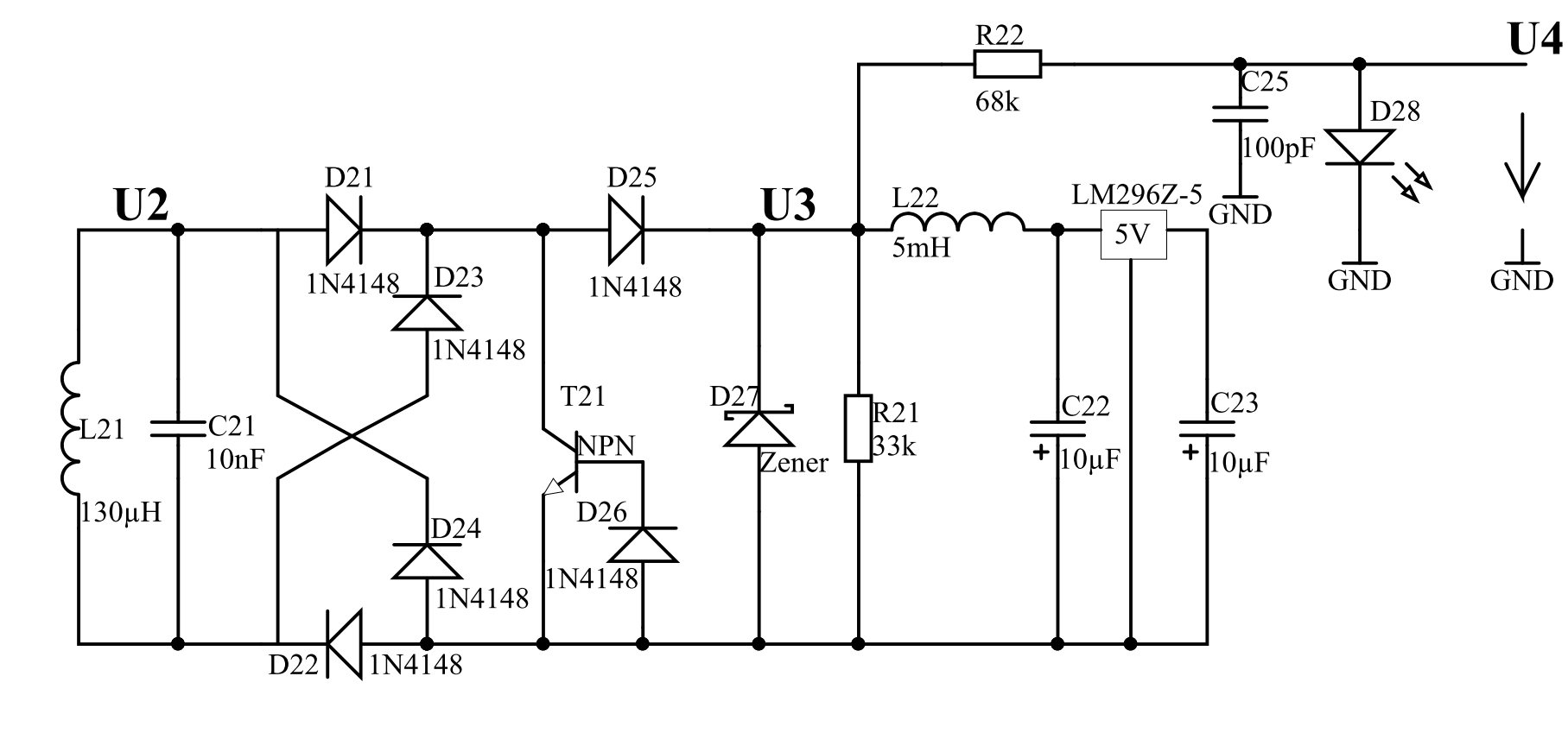


Abbildung 5. - Messpunkte Tag Primäreinspeisung

Am Tag wurde der µC ebenfalls entfernt, denn das empfangen Signal wird nur gemessen und nicht verarbeitet.

#### Messergebnisse

Die Messergebnisse wurden aus Übersichtlichkeit zusammengefasst. Das heißt das zusammengehörige Spannungen in einem Oszillogramm veranschaulicht wurden.

##### U1 & U2 (AM Trigger)

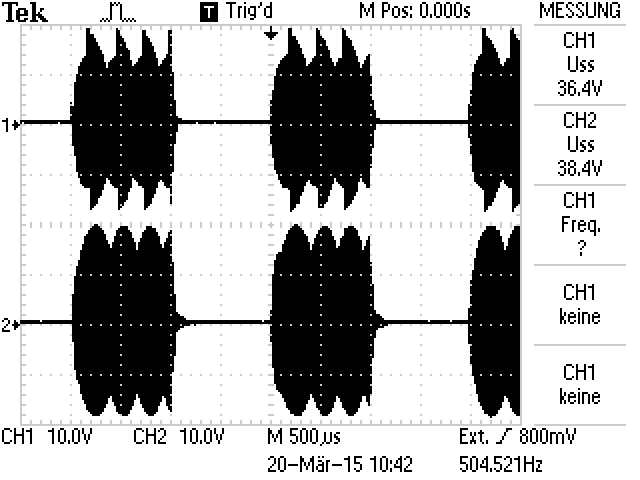


Abbildung 6. - U1 & U2 (AM Trigger)

Channel 1 stellt die Spannung U1 und Channel 2 die Spannung U2. Man sieht sehr deutlich, dass die Übertragung einwandfrei funktioniert. Es sind sehr ähnliche Spannungswerte vorhanden sowie die richtige Übertragungsfrequenz (vergl. Abb. 5.) Bei diesem Oszillogramm wurde auf die Amplitudenmodulation getriggert.

##### U1 & U2 (Normaler Trigger)

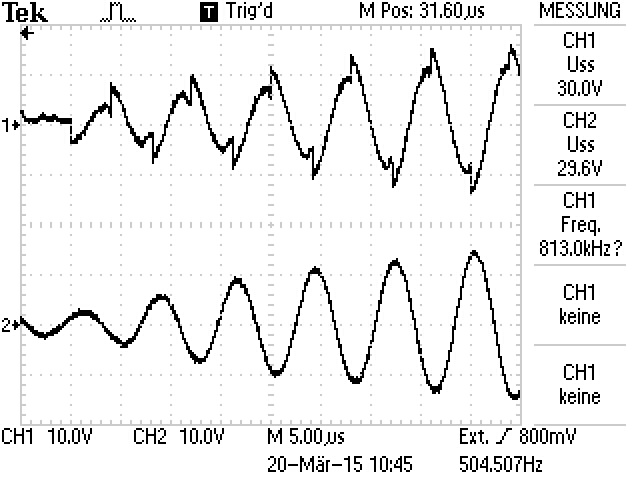


Abbildung 7. - U1 & U2 (Normaler Trigger)

Es ist hier das exakt selbe Signal zu sehen nur mit dem Unterschied, dass hier normal getriggert wurde. Dies wurde gemacht um mit dem Potentiometer P12 (vergl. Abb. 4) die Genauigkeit einzustellen.

#### U3

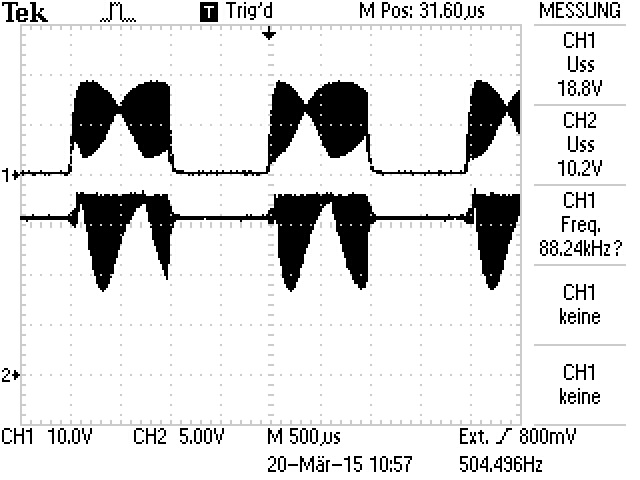
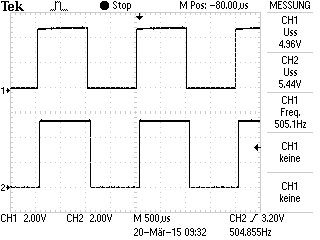
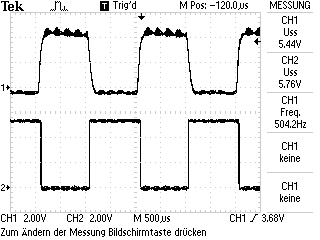
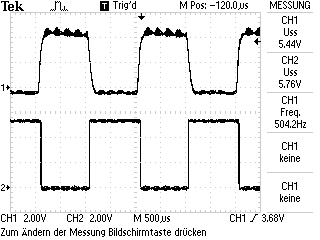
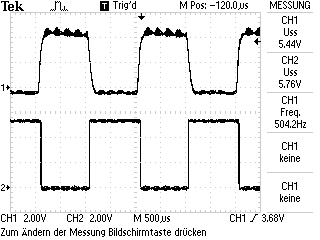


Abbildung 8. - Spannung an Diode 27

Channel 1 beschreibt die Spannung vor der Diode 27 (Zener) und Channel 2 danach.

#### U4 (UA)



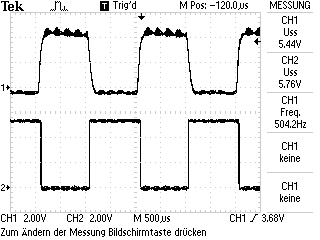


Abbildung 9. - Eingangssignal zu Ausgangssignal

Channel 1 zeigt das Eingangssignal und Channel 2 das Ausgangssignal. Die Übertragung sieht fast ident aus => Primärseitiges senden funktioniert einwandfrei.

#### Messpunkte

Alle Signale wurden mit einem Oszilloskop und Tastköpfen gemessen.

##### Reader

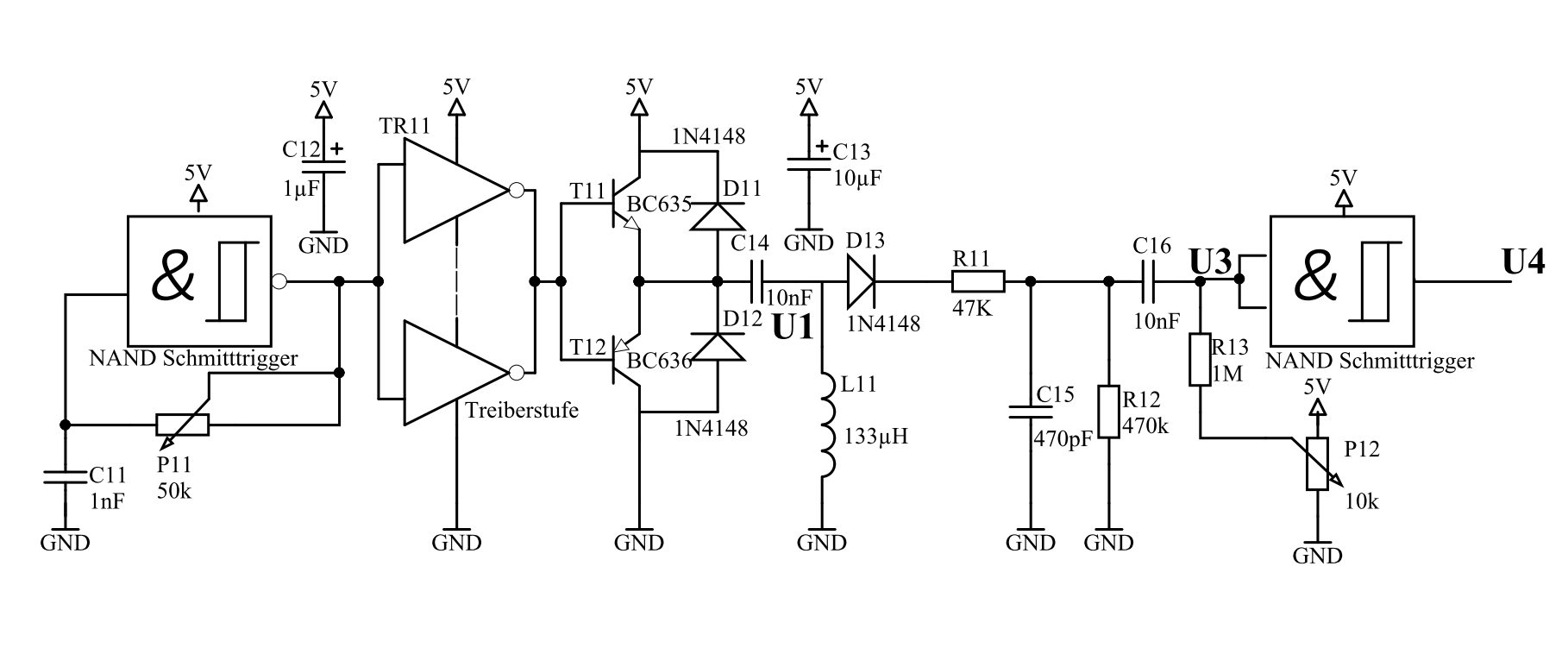


Abbildung 10. - Messpunkte Reader Sekundäreinspeisung

Bei der Sekundäreinspeisung wird die umgekehrte Übertragung getestet, weil schlussendlich die senden und empfangen auf beiden Seiten möglich sein soll. Es wurde der Funktionsgenerator am Eingang entfernt und am Tag angeschlossen.

##### Tag

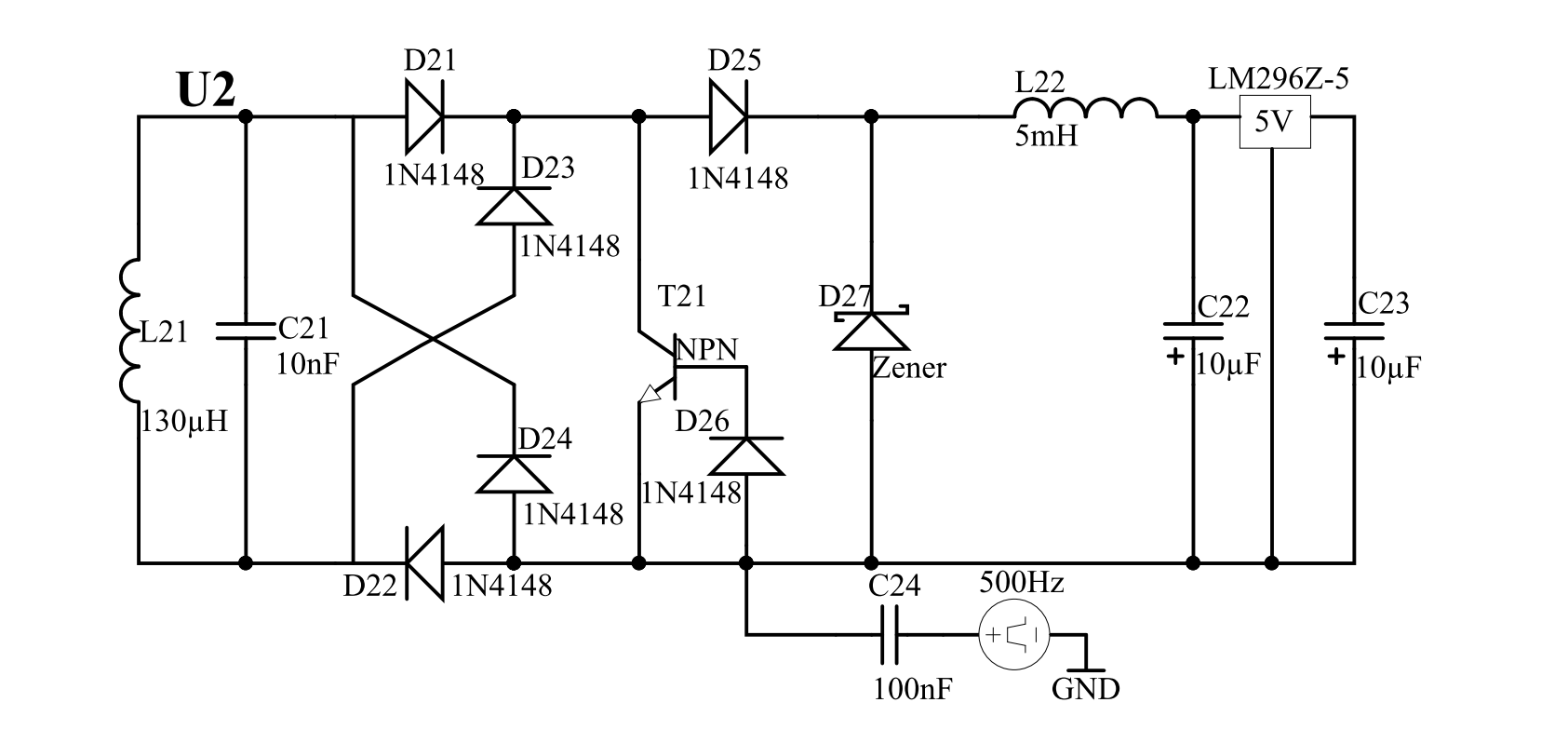


Abbildung 11. - Messpunkte Tag Sekundäreinspeisung

Der Tag sendet nun das Signal an den Reader. Zusätzlich wurde der Demodulator entfernt, weil er nicht gebraucht wird.

#### Messergebnisse

##### U1 & U2 (AM Trigger)

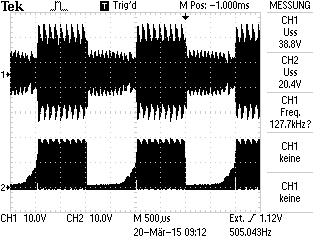


Abbildung 12. - U1 & U2 (AM Trigger)

Channel 1 misst die Primärseite und Channel 2 die Sekundärseite. Durch den Vollweggleichrichter geht der negative Teil des Signals verloren. Hier wurde auf die Amplitudenmodulation getriggert

##### U1 & U2 (Normaler Trigger)

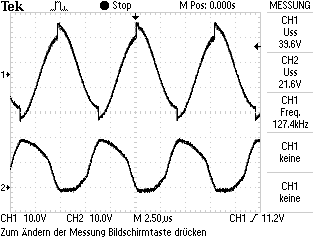


Abbildung 13. - U1 & U2 (Normaler Trigger)

Hier sieht man dasselbe Signal, nur wurde nicht auf die AM sondern auf das Signal selbst getriggert. Man kann bei Channel 2 bereits ein sehr schwaches Rechteck erkennen.

##### U3 & U4

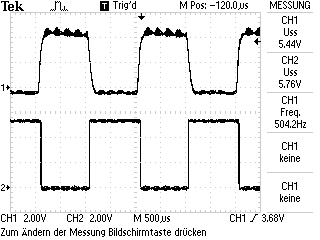


Abbildung 14. - U3 & U4 (Schmitttrigger)

Channel 1 zeigt das bereits geglättete Signal ohne Offset. Dieses hätte aber noch zu viele Störungen für einen µController, außerdem ist es noch invertiert. Deswegen wird es nochmal durch den Schmitttrigger geführt (Channel 2).

##### Eingangssignal & U4 (Ausgangssignal)

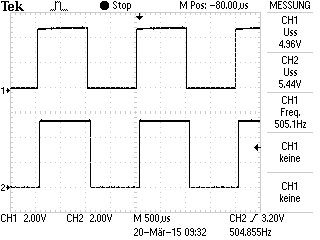


Abbildung 15. - Eingangssignal zu Ausgangssignal

Channel 1 zeigt das Eingangssignal des Funktionsgenerator und Channel 2 das Ausgangssignal (U4). Die beiden Signale sind ident => Die Hardware ist voll funktionstüchtig und es ist möglich mit den µControllern Signale zu übertragen. Die Schaltung kann jetzt betrieben werden wie in Abb. 1 und Abb. 2

# Software

Folgende Librarys bzw. Headerfiles wurden geschrieben.

## Librarys

### LCD.h

Beinhaltet alle notwendigen Funktionen zur kompletten Ansteuerung eines LCD Displays.

### Matrixtastatur.h

Beinhaltet eine Funktion welche bei Aufruf die gedrückte Taste zurückliefert.

unsigned char getTaste(void);

Mögliche Rückgabewerte sind alle möglichen Zeichen als Charakter oder 0, falls mehrere oder keine Taste gedrückt ist.

### Manchester.h

Beinhaltet globale Definitionen und Variablen welche sowohl für Sender als auch Empfänger verwendet wurden.

Für Tag und Reader wurden jeweils noch die Header-Files tag.h sowie reader.h erstellt.

## Programmlisting

### LCD.h

1 **|** //////////////////////////////////////////////////////////////////////////////

2 **|** //

3 **|** // Headerfile LCD.h zur LCD Ansteuerung am Port D

4 **|** //

5 **|** // Befehle: LCD\_init(), LCD\_cmd(char data), LCD\_send(char data)

6 **|** // LCD\_string(char \*data)

7 **|** //

8 **|** // LCD\_init(); initialisiert Port D

9 **|** // und LCD im 4-Bit Mode, 2 Zeilen, 5x7 Dots

10 **|** // Bsp.: LCD\_init();

11 **|** //

12 **|** // LCD\_cmd(char data); schickt Befehl ans LCD

13 **|** // Bsp.: LCD\_cmd(0xC5); //gehe zu 2. Zeile, 6. Position

14 **|** //

15 **|** // LCD\_send(char data); schickt Daten ans LCD

16 **|** // Bsp.: LCD\_send(0xEF); //sendet ein ö

17 **|** //

18 **|** // LCD\_string(char \*data); schickt eine Zeichenkette ans LCD

19 **|** // Bsp.: LCD\_string("Hallo"); //sendet Hallo

20 **|** //

21 **|** // Pinbelegung am Board:

22 **|** // LCD | Atmega16 | Bemerkung

23 **|** // ----|----------|--------------------

24 **|** // DB7 | PD3 |

25 **|** // DB6 | PD2 |

26 **|** // DB5 | PD1 |

27 **|** // DB4 | PD0 |

28 **|** // DB3 | - | wird nicht benötigt

29 **|** // DB2 | - | wird nicht benötigt

30 **|** // DB1 | - | wird nicht benötigt

31 **|** // DB0 | - | wird nicht benötigt

32 **|** // E | PD5 |

33 **|** // R/W | - | per HW auf GND gelegt

34 **|** // RS | PD4 |

35 **|** //

36 **|** ////////////////////////////////////////////////////////////////////////////////

37 **|**

38 **|** //-- Hier die Pinzuordnung bei Bedarf aendern (siehe Tabelle oben) --//

39 **|** #define DB7 PD7

40 **|** #define DB6 PD6

41 **|** #define DB5 PD5

42 **|** #define DB4 PD4

43 **|** #define E PD3

44 **|** #define RS PD2

45 **|**

46 **|** //------------------- Ende Pinzuordnung ------------------------------//

47 **|**

48 **|**

49 **|** #include **<**avr**/**io**.**h**>**

50 **|** #include **<**util**/**delay**.**h**>** // \_delay\_ms() geht nur bis max. 262.14 ms / F\_CPU

51 **|**

52 **|**

53 **|** void delay\_ms **(**unsigned int ms**)** //Hilfsfunktion: Zeitvernichtung

54 **|** **{**

55 **|** **for** **(**unsigned int i**=**0**;** i**<**ms**;** i**++)**

56 **|** **{**

57 **|** \_delay\_ms**(**1**);**

58 **|** **}**

59 **|**

60 **|** **}**

61 **|**

62 **|** void Enable**(**void**)** //Hilfsfunktion: H=>L Flanke der Enable Leitung (E)

63 **|** **{**

64 **|** PORTD **=** PORTD **|** **(**1**<<**E**);** //E = 1

65 **|** delay\_ms**(**5**);**

66 **|** PORTD **=** PORTD **&~(**1**<<**E**);** //E = 0

67 **|** delay\_ms**(**5**);**

68 **|** **}**

69 **|**

70 **|** ////////////////////////////////////////////////////////////////////////////

71 **|** //

72 **|** // LCD\_init(..) Initialisierung: Port D, 4-Bit Mode, 2 Zeilen, 5x7 Dots

73 **|** //

74 **|** ////////////////////////////////////////////////////////////////////////////

75 **|** void LCD\_init**(**void**)**

76 **|** **{**

77 **|** DDRD **=** DDRD **|(**1**<<**E**)** **|(**1**<<**RS**);** //E,RS als Ausgang

78 **|** DDRD **=** DDRD **|(**1**<<**DB7**)|(**1**<<**DB6**)|(**1**<<**DB5**)|(**1**<<**DB4**);** //DB7..DB4 als Ausgang

79 **|**

80 **|** delay\_ms**(**50**);** //lt. Datenblatt min. 15ms nach Power ON warten

81 **|** PORTD **=** PORTD **&** **(~(**1**<<**RS**)** **&** **~(**1**<<**E**));** //RS=0,E=0 (RW=0 per HW)

82 **|**

83 **|** // Function Set

84 **|** //DB7..DB4 = 0011

85 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**));** //Interface auf 8 Bit

86 **|** PORTD **=** PORTD **|** **(**1**<<**DB5**)** **|** **(**1**<<**DB4**);**

87 **|** Enable**();**

88 **|**

89 **|** //DB7..DB4 = 0011

90 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**));** //Interface auf 8 Bit

91 **|** PORTD **=** PORTD **|** **(**1**<<**DB5**)** **|** **(**1**<<**DB4**);**

92 **|** Enable**();**

93 **|**

94 **|** //DB7..DB4 = 0011

95 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**));** //Interface auf 8 Bit

96 **|** PORTD **=** PORTD **|** **(**1**<<**DB5**)** **|** **(**1**<<**DB4**);**

97 **|** Enable**();**

98 **|**

99 **|** //DB7..DB4 = 0010

100 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**)** **&~(**1**<<**DB4**));**

101 **|** PORTD **=** PORTD **|** **(**1**<<**DB5**);** //Interface auf 4 Bit

102 **|** Enable**();**

103 **|**

104 **|** // 2-zeilig, 5x8 Matrix //

105 **|** //DB7..DB4 = 0010

106 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**)** **&~(**1**<<**DB4**));**

107 **|** PORTD **=** PORTD **|** **(**1**<<**DB5**);** //Upper Nibble

108 **|** Enable**();**

109 **|**

110 **|** //DB7..DB4 = 1000

111 **|** PORTD **=** PORTD **|** **(**1**<<**DB7**);** //Lower Nibble

112 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB6**)** **&** **~(**1**<<**DB5**)** **&** **~(**1**<<**DB4**));**

113 **|** Enable**();**

114 **|**

115 **|** //Display Off //

116 **|** //DB7..DB4 = 0000

117 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**)** **&** **~(**1**<<**DB5**)** **&** **~(**1**<<**DB4**));** //Upper Nibble

118 **|** Enable**();**

119 **|**

120 **|** //DB7..DB4 = 1000

121 **|** PORTD **=** PORTD **|** **(**1**<<**DB7**);** //Lower Nibble

122 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB6**)** **&** **~(**1**<<**DB5**)** **&** **~(**1**<<**DB4**));**

123 **|** Enable**();**

124 **|**

125 **|** //Clear Display //

126 **|** //DB7..DB4 = 0000

127 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**)** **&** **~(**1**<<**DB5**)** **&** **~(**1**<<**DB4**));** //Upper Nibble

128 **|** Enable**();**

129 **|**

130 **|** //DB7..DB4 = 0001

131 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**)** **&** **~(**1**<<**DB5**));** //Lower Nibble

132 **|** PORTD **=** PORTD **|** **(**1**<<**DB4**);**

133 **|** Enable**();**

134 **|**

135 **|** //No Display Shift //

136 **|** //DB7..DB4 = 0000

137 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**)** **&** **~(**1**<<**DB5**)** **&** **~(**1**<<**DB4**));** //Upper Nibble

138 **|** Enable**();**

139 **|**

140 **|** //DB7..DB4 = 0011

141 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**));** //Lower Nibble

142 **|** PORTD **=** PORTD **|** **(**1**<<**DB5**)** **|** **(**1**<<**DB4**);**

143 **|** Enable**();**

144 **|**

145 **|** // Display ON , Cursor ON, Blinken ON //

146 **|** //DB7..DB4 = 0000

147 **|** PORTD **=** PORTD **&** **(~(**1**<<**DB7**)** **&** **~(**1**<<**DB6**)** **&** **~(**1**<<**DB5**)** **&** **~(**1**<<**DB4**));** //Upper Nibble

148 **|** Enable**();**

149 **|**

150 **|** //DB7..DB4 = 1111

151 **|** PORTD **=** PORTD **|** **(**1**<<**DB7**)** **|** **(**1**<<**DB6**)** **|** **(**1**<<**DB5**)** **|** **(**1**<<**DB4**);** //Lower Nibble

152 **|** Enable**();**

153 **|** **}**

154 **|**

155 **|** ////////////////////////////////////////////////////////////////////////////

156 **|** //

157 **|** // LCD\_send(..) sendet 1 Byte im 4-Bit Mode

158 **|** //

159 **|** ////////////////////////////////////////////////////////////////////////////

160 **|** void LCD\_send**(**char data**)**

161 **|** **{**

162 **|** char temp **=** data**;**

163 **|**

164 **|** PORTD **=** PORTD **|** **(**1**<<**RS**);** //SFR vom LCD mit RS auf Daten umschlten

165 **|**

166 **|** //Upper Nibble senden

167 **|** **if** **(**temp **&** 0b10000000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB7**);}**

168 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB7**);}**

169 **|**

170 **|** **if** **(**temp **&** 0b01000000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB6**);}**

171 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB6**);}**

172 **|**

173 **|** **if** **(**temp **&** 0b00100000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB5**);}**

174 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB5**);}**

175 **|**

176 **|** **if** **(**temp **&** 0b00010000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB4**);}**

177 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB4**);}**

178 **|**

179 **|** Enable**();**

180 **|** delay\_ms**(**1**);**

181 **|**

182 **|** //Lower Nibble senden

183 **|** **if** **(**temp **&** 0b00001000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB7**);}**

184 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB7**);}**

185 **|**

186 **|** **if** **(**temp **&** 0b00000100**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB6**);}**

187 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB6**);}**

188 **|**

189 **|** **if** **(**temp **&** 0b00000010**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB5**);}**

190 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB5**);}**

191 **|**

192 **|** **if** **(**temp **&** 0b00000001**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB4**);}**

193 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB4**);}**

194 **|**

195 **|** Enable**();**

196 **|** delay\_ms**(**1**);**

197 **|**

198 **|** **}**

199 **|**

200 **|** ////////////////////////////////////////////////////////////////////////////

201 **|** //

202 **|** // LCD\_cmd(..) Befehl senden im 4-Bit Mode

203 **|** //

204 **|** ////////////////////////////////////////////////////////////////////////////

205 **|** void LCD\_cmd**(**char data**)**

206 **|** **{**

207 **|** char temp **=** data**;**

208 **|**

209 **|** PORTD **=** PORTD **&** **~(**1**<<**RS**);** //SFR vom LCD mit RS auf Befehle umschalten

210 **|**

211 **|** //Upper Nibble senden

212 **|** **if** **(**temp **&** 0b10000000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB7**);}**

213 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB7**);}**

214 **|**

215 **|** **if** **(**temp **&** 0b01000000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB6**);}**

216 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB6**);}**

217 **|**

218 **|** **if** **(**temp **&** 0b00100000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB5**);}**

219 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB5**);}**

220 **|**

221 **|** **if** **(**temp **&** 0b00010000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB4**);}**

222 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB4**);}**

223 **|**

224 **|** Enable**();**

225 **|** delay\_ms**(**1**);**

226 **|**

227 **|** //Lower Nibble senden

228 **|** **if** **(**temp **&** 0b00001000**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB7**);}**

229 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB7**);}**

230 **|**

231 **|** **if** **(**temp **&** 0b00000100**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB6**);}**

232 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB6**);}**

233 **|**

234 **|** **if** **(**temp **&** 0b00000010**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB5**);}**

235 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB5**);}**

236 **|**

237 **|** **if** **(**temp **&** 0b00000001**)** **{**PORTD **=** PORTD **|** **(**1**<<**DB4**);}**

238 **|** **else** **{**PORTD **=** PORTD **&** **~(**1**<<**DB4**);}**

239 **|**

240 **|** Enable**();**

241 **|** delay\_ms**(**1**);**

242 **|**

243 **|** **}**

244 **|**

245 **|** ////////////////////////////////////////////////////////////////////////////

246 **|** //

247 **|** // LCD\_string(..) sendet ganzen String im 4-Bit Mode

248 **|** //

249 **|** ////////////////////////////////////////////////////////////////////////////

250 **|** void LCD\_string**(**char **\***data**)**

251 **|** **{**

252 **|** **while** **(\***data **!=** '\0'**)** //bis zum letzten Zeichen

253 **|** **{**LCD\_send**(\***data**++);}**

254 **|** **}**

### Matrixtastatur.h

1 **|** #include **<**avr**/**io**.**h**>** // Include File fr IO Definitionen

2 **|** #include **<**avr**/**interrupt**.**h**>** // Include File fr sei(), cli()

3 **|** #include **<**util**/**delay**.**h**>**

4 **|**

5 **|** // PD0 PD1 PD2 PD3

6 **|** // | | | |

7 **|** //

8 **|** // 1 2 3 A -- PD4 Ausgang

9 **|** // 4 5 6 B -- PD5 Ausgang

10 **|** // 7 8 9 C -- PD6 Ausgang

11 **|** // \* 0 # A -- PD7 Ausgang

12 **|**

13 **|** #define PORT\_USED PORTB

14 **|** #define DDR\_USED DDRB

15 **|** #define PIN\_USED PINB

16 **|**

17 **|** unsigned char getTaste**(**void**)**

18 **|** **{**

19 **|** char zeile**,**spalte**;**

20 **|** // \*\*\*\*\*\*\*\*\*\*\* Auswertung der Zeile \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

21 **|** DDR\_USED **=** 0xF0**;** //Zwischenschritt lt. Dantenblatt

22 **|** PORT\_USED **=** 0xF0**;** //von INPUT mit Pull Up auf OUTPUT LOW

23 **|** DDR\_USED **=** 0x0F**;** //PD 4-7 als OUTPUT (Spalten), PB 3-0 als INPUT (Zeilen)

24 **|** PORT\_USED **=** PORT\_USED **|** 0xF0**;** //PD 3-0 int. Pull Up Widerstnde ein

25 **|** PORT\_USED **=** PORT\_USED **&** 0xF0**;** //PD 4-7 auf "0"

26 **|**

27 **|** **switch** **((**PIND **&** 0xF0**))**

28 **|** **{** // Spalte Zeile

29 **|** **case** 0xE0**:** //Zeile PD0 = "0" 0000 1110

30 **|** spalte **=** 1**;** **break;**

31 **|** **case** 0xD0**:** //Zeile PD1 = "0" 0000 1101

32 **|** spalte **=** 2**;** **break;**

33 **|** **case** 0xB0**:** //Zeile PD2 = "0" 0000 1011

34 **|** spalte **=** 3**;** **break;**

35 **|** **case** 0x70**:** //Zeile PD3 = "0" 0000 0111

36 **|** spalte **=** 4**;** **break;**

37 **|** **default** **:** //mehrere Tasten wurden gedrckt

38 **|** spalte **=** 0**;** **break;**

39 **|** **}**

40 **|**

41 **|** // \*\*\*\*\*\*\*\*\*\*\*\* Auswertung der Spalten \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

42 **|** DDR\_USED **=** 0x0F**;** //Zwischenschritt lt. Dantenblatt für Umkonfiguration

43 **|** PORT\_USED **=** 0x0F**;** //von INPUT mit Pull Up auf OUTPUT LOW

44 **|** DDR\_USED **=** 0xF0**;** //PB 4-7 als INPUT (Spalten), PD3-0 als OUTPUT (Zeilen)

45 **|** PORT\_USED **=** **(**spalte **|** 0x0F**);** //Erg. von Zeile auf PD3-0 damit gedr. //Zeile=0 und alle anderen Zeilen=1

46 **|**

47 **|** //PD 4-7 int. Pull Up Widerstnde ein

48 **|** **switch** **(**PIN\_USED **&** 0x0F**)** // oberen 4 Bit = 1 => gedr. Taste zieht dann //Spalte von 1=>0

49 **|** **{** // Spalte Zeile

50 **|** **case** 0x0E**:** //Spalte PD4 = "0" 1110 0000

51 **|** zeile **=** 1**;** **break;**

52 **|** **case** 0x0D**:** //Spalte PD5 = "0" 1101 0000

53 **|** zeile **=** 2**;** **break;**

54 **|** **case** 0x0B**:** //Spalte PD6 = "0" 1011 0000

55 **|** zeile **=** 3**;** **break;**

56 **|** **case** 0x07**:** //Spalte PD7 = "0" 0111 0000

57 **|** zeile **=** 4**;** **break;**

58 **|** **default:** //mehrere Spalten sind auf "0"

59 **|** zeile **=** 0**;** **break;**

60 **|** **}**

61 **|**

62 **|** **if** **(**zeile **==** 1 **&&** spalte **==** 1**)** **return** '1'**;**

63 **|** **else** **if** **(**zeile **==** 1 **&&** spalte **==** 2**)** **return** '2'**;**

64 **|** **else** **if** **(**zeile **==** 1 **&&** spalte **==** 3**)** **return** '3'**;**

65 **|** **else** **if** **(**zeile **==** 1 **&&** spalte **==** 4**)** **return** 'A'**;**

66 **|** **else** **if** **(**zeile **==** 2 **&&** spalte **==** 1**)** **return** '4'**;**

67 **|** **else** **if** **(**zeile **==** 2 **&&** spalte **==** 2**)** **return** '5'**;**

68 **|** **else** **if** **(**zeile **==** 2 **&&** spalte **==** 3**)** **return** '6'**;**

69 **|** **else** **if** **(**zeile **==** 2 **&&** spalte **==** 4**)** **return** 'B'**;**

70 **|** **else** **if** **(**zeile **==** 3 **&&** spalte **==** 1**)** **return** '7'**;**

71 **|** **else** **if** **(**zeile **==** 3 **&&** spalte **==** 2**)** **return** '8'**;**

72 **|** **else** **if** **(**zeile **==** 3 **&&** spalte **==** 3**)** **return** '9'**;**

73 **|** **else** **if** **(**zeile **==** 3 **&&** spalte **==** 4**)** **return** 'C'**;**

74 **|** **else** **if** **(**zeile **==** 4 **&&** spalte **==** 1**)** **return** '\*'**;**

75 **|** **else** **if** **(**zeile **==** 4 **&&** spalte **==** 2**)** **return** '0'**;**

76 **|** **else** **if** **(**zeile **==** 4 **&&** spalte **==** 3**)** **return** '#'**;**

77 **|** **else** **if** **(**zeile **==** 4 **&&** spalte **==** 4**)** **return** 'D'**;**

78 **|** **return** 0**;**

79 **|** **}**

### Manchester.h

1 **|** //

2 **|** // Manchester.h

3 **|** //

4 **|**

5 **|** #ifndef \_ManchesterGlobal\_h

6 **|** #define \_ManchesterGlobal\_h

7 **|**

8 **|** #define F\_CPU 8000000UL

9 **|**

10 **|** /\* 2500 Bits pro Sekunde\*/

11 **|** #define BAUD\_RATE 2500

12 **|**

13 **|** /\* Die Zeit für ein Bit pro Sekunde \*/

14 **|** #define BIT\_TIME 400

15 **|**

16 **|** #define HALF\_TIME 200

17 **|**

18 **|** /\* DIe Zeit zwischen zwei Samples in µs/cycles \*/

19 **|** #define SAMPLE\_TIME 100

20 **|**

21 **|** #define DATA\_SAMPLES 32

22 **|**

23 **|** #define SET\_BIT**(**byte**,** bit**)** **((**byte**)** **|=** **(**1UL **<<** **(**bit**)))**

24 **|**

25 **|** #define CLEAR\_BIT**(**byte**,**bit**)** **((**byte**)** **&=** **~(**1UL **<<** **(**bit**)))**

26 **|**

27 **|** #define IS\_SET**(**byte**,**bit**)** **(((**byte**)** **&** **(**1UL **<<** **(**bit**)))** **>>** **(**bit**))**

28 **|**

29 **|** #endif

## Sende-Empfänger (Reader)

Verwendete Librarys: LCD.h und Manchester.h

### C-File

1 **|** #include **<**avr**/**io**.**h**>**

2 **|** #include **<**util**/**delay**.**h**>**

3 **|** #include **<**avr**/**interrupt**.**h**>**

4 **|**

5 **|** #include **<**stdio**.**h**>**

6 **|** #include **<**stdint**.**h**>**

7 **|**

8 **|** #include "../Global/Manchester.h"

9 **|** #include "../Global/LCD.h"

10 **|**

11 **|** #include "reader.h"

12 **|**

13 **|** volatile uint32\_t samples **=** 0**;**

14 **|** volatile uint8\_t samplesReady **=** 0**;**

15 **|** volatile int8\_t sampleCount **=** DATA\_SAMPLES **-** 1**;**

16 **|** uint8\_t receiving **=** 0**,** data **=** 0**,** connectionCount **=** 0**;**

17 **|**

18 **|** int main**(**void**)**

19 **|** **{**

20 **|**

21 **|** struct Queue q **=** **{.**size **=** 0**};**

22 **|** uint16\_t chksm**;**

23 **|**

24 **|** LCD\_init**();**

25 **|** initRX**();**

26 **|** startRX**();**

27 **|**

28 **|** **while(**1**)**

29 **|** **{**

30 **|** **if** **(**receiving **&&** samplesReady **&&** interpretSamples**(**getSamples**()))**

31 **|** **{**

32 **|** pushQ**(&**q**,**data**);**

33 **|**

34 **|** **if** **(**q**.**size **==** 14**)** **break;**

35 **|** **}**

36 **|** **}**

37 **|**

38 **|** stopRX**();**

39 **|**

40 **|** chksm **=** checksumme**(**q**.**data**,**12**);**

41 **|**

42 **|** CLEAR\_BIT**(**LED\_PORT**,**LED**);**

43 **|**

44 **|** **if** **(**chksm **==** **((**q**.**data**[**12**]** **<<** 8**)** **|** q**.**data**[**13**]))** // daten OK, checksumme ok

45 **|** SET\_BIT**(**LED\_PORT**,**LED**);**

46 **|**

47 **|** LCD\_string**(**q**.**data**);**

48 **|**

49 **|** **return** 0**;**

50 **|**

51 **|** **}**

52 **|**

53 **|** int8\_t popQ**(**struct Queue**\*** queue**)**

54 **|** **{**

55 **|** unsigned char i **=** 0**,** j **=** 1**;**

56 **|** char item **=** queue**->**data**[**0**];**

57 **|**

58 **|** /\* Alle Elemente ins strukt \*/

59 **|** **for** **(;** j **<** queue**->**size**;** **++**j**,** **++**i**)**

60 **|** **{** queue**->**data**[**i**]** **=** queue**->**data**[**j**];** **}**

61 **|**

62 **|** /\* Letztes Element löschen \*/

63 **|** queue**->**data**[--**queue**->**size**]** **=** 0**;**

64 **|**

65 **|** **return** item**;**

66 **|** **}**

67 **|**

68 **|** void pushQ**(**struct Queue**\*** queue**,** const uint8\_t c**)**

69 **|** **{**

70 **|** **if** **(**queue**->**size **<** BUFFER\_SIZE**)**

71 **|** **{** queue**->**data**[**queue**->**size**++]** **=** c**;** **}**

72 **|** **}**

73 **|**

74 **|** uint16\_t checksumme**(**const uint8\_t**\*** data**,** uint16\_t bytes**)**

75 **|** **{**

76 **|**

77 **|** uint16\_t summe1 **=** 0xFF**;**

78 **|** uint16\_t summe2 **=** 0xFF**;**

79 **|**

80 **|** uint16\_t laenge**;**

81 **|**

82 **|** **while** **(**bytes**)**

83 **|** **{**

84 **|** /\* Maximale Länge bevor Overflow ensteht \*/

85 **|** laenge **=** **(**bytes **>** 20**)** **?** 20 **:** bytes**;**

86 **|**

87 **|** bytes **-=** laenge**;**

88 **|**

89 **|** **do**

90 **|** **{**

91 **|** summe1 **+=** **\*(**data**++);**

92 **|** summe2 **+=** summe1**;**

93 **|** **}**

94 **|**

95 **|** **while** **(--**laenge**);**

96 **|**

97 **|** summe1 **=** **(**summe1 **&** 0xFF**)** **+** **(**summe1 **>>** 8**);**

98 **|** summe2 **=** **(**summe2 **&** 0xFF**)** **+** **(**summe2 **>>** 8**);**

99 **|** **}**

100 **|**

101 **|** /\* Zur Sicherheit unnötiges abschneiden \*/

102 **|** summe1 **=** **(**summe1 **&** 0xFF**)** **+** **(**summe1 **>>** 8**);**

103 **|** summe2 **=** **(**summe2 **&** 0xFF**)** **+** **(**summe2 **>>** 8**);**

104 **|**

105 **|** /\* Beiden Summen als 16 Bit Wert zurückgeben\*/

106 **|** **return** summe2 **<<** 8 **|** summe1**;**

107 **|** **}**

108 **|**

109 **|** uint32\_t getSamples**(**void**)**

110 **|** **{**

111 **|** /\* Flag zurücksezen \*/

112 **|** samplesReady **=** 0**;**

113 **|**

114 **|** **return** samples**;**

115 **|** **}**

116 **|**

117 **|** void initRX**(**void**)**

118 **|** **{**

119 **|** /\* RX Pin als Input setzen \*/

120 **|** CLEAR\_BIT**(**RX\_DDR**,**RX\_PIN**);**

121 **|**

122 **|** /\* LED Pin als Ausgang setzen \*/

123 **|** SET\_BIT**(**LED\_DDR**,**LED**);**

124 **|**

125 **|** /\* Enable pin change interrupt \*/

126 **|** PCICR **|=** **(**1**<<**PCIE0**);**

127 **|**

128 **|** /\* Pin change ISR auf PCINT1 (PB1) \*/

129 **|** SET\_BIT**(**PCMSK0**,**PCINT1**);**

130 **|**

131 **|** /\* Timer1 mit clk/4096 prescaler einschalten \*/

132 **|** TCCR1A **|=** 0x0D**;**

133 **|**

134 **|** /\* Everflow interrupt für Timer1 \*/

135 **|** SET\_BIT**(**TIMSK0**,**TOIE1**);**

136 **|**

137 **|** /\* Sample alle 100 µs \*/

138 **|** OCR0A **=** SAMPLE\_TIME**;**

139 **|**

140 **|** /\* output compare interrupt für Timer0 \*/

141 **|** SET\_BIT**(**TIMSK0**,**OCIE0A**);**

142 **|**

143 **|** /\* Timer0 mit clk/8 prescaler, ergibt 1µs pro zyklus, -> bei 8Mhz.

144 | Timer0 wird benutzt zum samplen \*/

145 **|** SET\_BIT**(**TCCR0B**,**CS01**);**

146 **|** **}**

147 **|**

148 **|** void startRX**(**void**)**

149 **|** **{**

150 **|** uint8\_t preambleBit**,** lows **=** 0**,** highs **=** 0**;**

151 **|**

152 **|** SET\_BIT**(**LED\_PORT**,**LED**);**

153 **|**

154 **|** /\* ISR gloabl freigeben\*/

155 **|** asm**(**"sei"**);**

156 **|**

157 **|** /\* Zurücksetzen \*/

158 **|** TCNT0 **=** TCNT1 **=** 0**;**

159 **|**

160 **|** /\* Einschalten während SYNC, wird 0 wenn die datenübetragung aus ist\*/

161 **|** receiving **=** 1**;**

162 **|**

163 **|** /\* Individuelle Bits zählen \*/

164 **|** sampleCount **=** 3**;**

165 **|**

166 **|** samples **=** 0**;**

167 **|**

168 **|** /\* so lange daten empfangen werden \*/

169 **|** **while** **(**receiving**)**

170 **|** **{**

171 **|** **if** **(**samplesReady**)**

172 **|** **{**

173 **|** /\* bit abfragen \*/

174 **|** preambleBit **=** getSamples**();**

175 **|**

176 **|** sampleCount **=** 3**;**

177 **|**

178 **|** /\* Die Präabmel enthällt 6 low bits (10) und 2 high bits (01), alles andere setzt den counter wieder zurück auf 0 \*/

179 **|** **if** **(**preambleBit **==** LOW**)**

180 **|** **{**

181 **|** **if** **(!**highs**)**

182 **|** **++**lows**;**

183 **|**

184 **|** **else**

185 **|** lows **=** highs **=** 0**;**

186 **|** **}**

187 **|**

188 **|** **else** **if** **(**preambleBit **==** HIGH**)**

189 **|** **{**

190 **|** **if** **(**lows **>=** 6**)**

191 **|** **{**

192 **|** **if** **(++**highs **>=** 2**)**

193 **|** **break;**

194 **|** **}**

195 **|**

196 **|** **else**

197 **|** lows **=** highs **=** 0**;**

198 **|** **}**

199 **|**

200 **|** **else**

201 **|** **{**

202 **|** lows **=** highs **=** 0**;**

203 **|** sampleCount **=** 2**;**

204 **|** **}**

205 **|** **}**

206 **|** **}**

207 **|**

208 **|** sampleCount **=** DATA\_SAMPLES **-** 1**;**

209 **|** **}**

210 **|**

211 **|**

212 **|** void stopRX**(**void**)**

213 **|** **{**

214 **|** receiving **=** 0**;**

215 **|**

216 **|** /\* Timer/Counter aussschalten\*/

217 **|** CLEAR\_BIT**(**TIMSK0**,**OCIE0A**);**

218 **|** CLEAR\_BIT**(**LED\_PORT**,**LED**);**

219 **|** **}**

220 **|**

221 **|** uint8\_t interpretSamples**(**const uint32\_t samps**)**

222 **|** **{**

223 **|** int8\_t i **=** 7**;**

224 **|**

225 **|** uint8\_t bit**;**

226 **|**

227 **|** **for** **(;** i **>=** 0**;** **--**i**)**

228 **|** **{**

229 **|** /\* Aktuelles Bit holen \*/

230 **|** bit **=** **(**samps **>>** **(**i**\***4**))** **&** 0x0F**;**

231 **|**

232 **|** **if** **(**bit **==** HIGH**)**

233 **|** SET\_BIT**(**data**,**i**);**

234 **|**

235 **|** **else** **if** **(**bit **==** LOW**)**

236 **|** CLEAR\_BIT**(**data**,**i**);**

237 **|**

238 **|** **else** **return** 0**;**

239 **|** **}**

240 **|**

241 **|** **return** 1**;**

242 **|** **}**

243 **|**

244 **|** ISR**(**TIMER1\_OVF\_vect**)**

245 **|** **{**

246 **|** **if** **(++**connectionCount **>=** CONNECTION\_TIME**)**

247 **|** **{**

248 **|** connectionCount **=** 0**;**

249 **|**

250 **|** stopRX**();**

251 **|** **}**

252 **|** **}**

253 **|**

254 **|** ISR**(**PCINT0\_vect**)**

255 **|** **{**

256 **|** /\* Timer setzen, was er bei einem pin change sein soll, 52 astatt 50 wegen ISR Aufruf Overhead\*/

257 **|** TCNT0 **=** 52**;**

258 **|** **}**

259 **|**

260 **|** ISR**(**TIMER0\_COMPA\_vect**)**

261 **|** **{**

262 **|** /\* Wenn aktuelles Sample hight ausliest -> aktuelles bit auf HIGH \*/

263 **|** **if** **(**IS\_SET**(**RX\_PORT**,**RX\_PIN**))**

264 **|** **{**

265 **|** /\* Reset timer now, after sampling \*/

266 **|** TCNT0 **=** 1**;**

267 **|**

268 **|** SET\_BIT**(**samples**,**sampleCount**);**

269 **|** **}**

270 **|**

271 **|** /\* sonst aktuelles bit auf LOW setzen \*/

272 **|** **else**

273 **|** **{**

274 **|** /\* timer nach sampling zurücksetzen \*/

275 **|** TCNT0 **=** 1**;**

276 **|**

277 **|** CLEAR\_BIT**(**samples**,**sampleCount**);**

278 **|** **}**

279 **|**

280 **|** /\* wenn bit fertig ist -> samplesReady flag setzen \*/

281 **|** **if** **(!** sampleCount**--)**

282 **|** **{**

283 **|** samplesReady **=** 1**;**

284 **|** sampleCount **=** DATA\_SAMPLES **-** 1**;**

285 **|** **}**

286 **|** **}**

### Header-File

1 **|**

2 **|** #ifndef \_\_ManchesterRX\_\_

3 **|** #define \_\_ManchesterRX\_\_

4 **|**

5 **|** #include **<**stdint**.**h**>**

6 **|**

7 **|** #define RX\_DDR DDRB

8 **|** #define RX\_PORT PINB

9 **|** #define RX\_PIN PB3

10 **|**

11 **|** #define LED PD7

12 **|** #define LED\_PORT PORTD

13 **|** #define LED\_DDR DDRD

14 **|**

15 **|** #define HIGH 0b0011

16 **|** #define LOW 0b1100

17 **|**

18 **|** #define CONNECTION\_TIME 77

19 **|**

20 **|** #define BUFFER\_SIZE 128

21 **|**

22 **|** struct Queue

23 **|** **{**

24 **|** uint8\_t data**[**BUFFER\_SIZE**];**

25 **|**

26 **|** uint8\_t size**;**

27 **|** **};**

28 **|**

29 **|** int8\_t popQ**(**struct Queue**\*** queue**);**

30 **|**

31 **|** void pushQ**(**struct Queue**\*** queue**,** const uint8\_t c**);**

32 **|**

33 **|** uint8\_t interpretSamples**(**const uint32\_t samps**);**

34 **|**

35 **|** void initRX**(**void**);**

36 **|**

37 **|** void startRX**(**void**);**

38 **|** void stopRX**(**void**);**

39 **|**

40 **|** uint32\_t getSamples**(**void**);**

41 **|**

42 **|** uint16\_t checksumme**(**const uint8\_t**\*** data**,** uint16\_t bytes**);**

43 **|** // Zum Berechnen der Checksumme

44 **|**

45 **|** #endif

## Tag

Verwendete Librarys: Matrixtastatur.h und Manchester.h

### C-File

1 **|** #include **<**avr**/**io**.**h**>**

2 **|** #include **<**avr**/**interrupt**.**h**>**

3 **|** #include **<**util**/**delay**.**h**>**

4 **|**

5 **|**

6 **|** #include "../Global/Manchester.h"

7 **|** #include "../Global/Matrixtastatur.h"

8 **|**

9 **|** #include "tag.h"

10 **|**

11 **|** int main**(**void**)**

12 **|** **{**

13 **|** unsigned char msg **[]** **=** "Text: "**;**

14 **|** initTag**();**

15 **|**

16 **|** unsigned char taste**;**

17 **|**

18 **|** **while(**1**)**

19 **|** **{**

20 **|** taste **=** getTaste**();**

21 **|** **if(**taste **!=** 0**)**

22 **|** sendeDaten**(**taste**,**1**);**

23 **|**

24 **|** **}**

25 **|**

26 **|** **return** 0**;**

27 **|** **}**

28 **|**

29 **|** void initTag**(**void**)**

30 **|** **{**

31 **|** /\* TX\_PIN als Ausgang setzen \*/

32 **|** SET\_BIT**(**TX\_DDR**,**TX\_PIN**);**

33 **|**

34 **|** /\* TCNT0 mit clk/8 prescaler einstellen bzw. einschalten \*/

35 **|** SET\_BIT**(**TCCR0B**,**CS01**);**

36 **|** **}**

37 **|**

38 **|** uint16\_t checksumme**(**const uint8\_t**\*** data**,** uint16\_t bytes**)**

39 **|** **{**

40 **|**

41 **|** uint16\_t summe1 **=** 0xFF**;**

42 **|** uint16\_t summe2 **=** 0xFF**;**

43 **|**

44 **|** uint16\_t laenge**;**

45 **|**

46 **|** **while** **(**bytes**)**

47 **|** **{**

48 **|** /\* Maximale Länge bevor Overflow ensteht \*/

49 **|** laenge **=** **(**bytes **>** 20**)** **?** 20 **:** bytes**;**

50 **|**

51 **|** bytes **-=** laenge**;**

52 **|**

53 **|** **do**

54 **|** **{**

55 **|** summe1 **+=** **\*(**data**++);**

56 **|** summe2 **+=** summe1**;**

57 **|** **}**

58 **|**

59 **|** **while** **(--**laenge**);**

60 **|**

61 **|** summe1 **=** **(**summe1 **&** 0xFF**)** **+** **(**summe1 **>>** 8**);**

62 **|** summe2 **=** **(**summe2 **&** 0xFF**)** **+** **(**summe2 **>>** 8**);**

63 **|** **}**

64 **|**

65 **|** /\* Zur Sicherheit unnötiges abschneiden \*/

66 **|** summe1 **=** **(**summe1 **&** 0xFF**)** **+** **(**summe1 **>>** 8**);**

67 **|** summe2 **=** **(**summe2 **&** 0xFF**)** **+** **(**summe2 **>>** 8**);**

68 **|**

69 **|** /\* Beiden Summen als 16 Bit Wert zurückgeben\*/

70 **|** **return** summe2 **<<** 8 **|** summe1**;**

71 **|** **}**

72 **|**

73 **|** void sendeDaten**(**const uint8\_t**\*** data**,** uint16\_t bytes**)**

74 **|** **{**

75 **|** uint8\_t i**;**

76 **|** uint16\_t chksm**;**

77 **|**

78 **|** /\* Checksumme der Daten berechnen \*/

79 **|** chksm **=** checksumme**(**data**,**bytes**);**

80 **|**

81 **|** /\* Viele 10s senden um am Empfänger aufs Signal synchronisieren zu können \*/

82 **|** **for(**i **=** 0**;** i **<** 75**;** **++**i**)**

83 **|** sendeByte**(**0**);**

84 **|**

85 **|** /\* Letzte 10s senden und dann zwei 01 Start Impulse \*/

86 **|** sendeByte**(**3**);**

87 **|**

88 **|** /\* Daten Byte für Byte senden \*/

89 **|** **for** **(**i **=** 0**;** i **<** bytes**;** **++**i**)**

90 **|** sendeByte**(**data**[**i**]);**

91 **|**

92 **|** /\* Checksumme anhängen, als erstrs oberes dann unteres byte \*/

93 **|** sendeByte**((**chksm **>>** 8**));**

94 **|** sendeByte**((**chksm **&** 0xFF**));**

95 **|**

96 **|** /\* Es werden keine Bits mehr gesendet, aber das letzte Bit muss noch fertig werden, sonst,

97 | wenn das letzte Bit HIGH ist, würde es nicht mehr erkannt werden, weil der TX Pin abgedreht

98 | wird nachher -> würde sonst eine 0 werden\*/

99 **|**

100 **|** **while(**TCNT0 **<** HALF\_TIME**);**

101 **|**

102 **|** /\* Sende Pin anschließend auf LOW setzen \*/

103 **|** CLEAR\_BIT**(**TX\_PORT**,**TX\_PIN**);**

104 **|** **}**

105 **|**

106 **|** void sendeByte**(**const uint8\_t byte**)**

107 **|** **{**

108 **|** int8\_t bit **=** 7**;**

109 **|**

110 **|** **for** **(;** bit **>=** 0**;** **--**bit**)**

111 **|** **{**

112 **|** **if** **(** IS\_SET**(**byte**,**bit**)** **)**

113 **|** sendeHIGHBit**(**1**);**

114 **|**

115 **|** **else**

116 **|** sendeLOWBit**(**1**);**

117 **|**

118 **|** **}**

119 **|**

120 **|** **}**

121 **|** void sendeHIGHBit**(**unsigned char anzahl**)**

122 **|** /\* Setze Pin auf LOW, warte auf die Mitte der Periode,

123 | \* dann wieder HIGH setzen und auf das Ende der Periode warten.

124 | \* => HIGH Bit laut Manchester Code (01)

125 | \*/

126 **|** **{**

127 **|** **for(**int i**=**0**;**i**<**anzahl**;**i**++)**

128 **|** **{**

129 **|** **while(**TCNT0 **<** **(**BIT\_TIME**/**2**));** // warten

130 **|** CLEAR\_BIT**(**TX\_PORT**,**TX\_PIN**);** // Sende PIN auf 0 setzen

131 **|** TCNT0 **=** 0**;**

132 **|**

133 **|** **while(**TCNT0 **<** **(**BIT\_TIME**/**2**));**

134 **|** SET\_BIT**(**TX\_PORT**,**TX\_PIN**);** // Sende PIN auf 1 setzen

135 **|** TCNT0 **=** 1**;**

136 **|**

137 **|** **}**

138 **|** **}**

139 **|**

140 **|** void sendeLOWBit**(**unsigned char anzahl**)**

141 **|** /\* Setze auf Pin auf HIGH, warte auf die Mitter der Periode,

142 | \* dann wieder LOW setzen und auf das Ende der Periode warten.

143 | \* => LOW Bit laut Manchester Code (10)

144 | \*/

145 **|** **{**

146 **|** **for(**int i**=**0**;**i**<**anzahl**;**i**++)**

147 **|** **{**

148 **|** **while(**TCNT0 **<** **(**BIT\_TIME**/**2**));** // warten

149 **|** SET\_BIT**(**TX\_PORT**,**TX\_PIN**);** // Sende PIN auf 1 setzen

150 **|** TCNT0 **=** 0**;**

151 **|**

152 **|** **while(**TCNT0 **<** **(**BIT\_TIME**/**2**));**

153 **|** CLEAR\_BIT**(**TX\_PORT**,**TX\_PIN**);** // Sende PIN auf 0 setzen

154 **|** TCNT0 **=** 1**;**

155 **|** **}**

156 **|** **}**

### Header-File

1 **|** #ifndef \_Manchester\_h

2 **|** #define \_Manchester\_h

3 **|**

4 **|** #include **<**stdint**.**h**>**

5 **|**

6 **|** #define TX\_PORT PORTB

7 **|** #define TX\_DDR DDRB

8 **|** #define TX\_PIN PB0

9 **|**

10 **|** void initTag**(**void**);**

11 **|**

12 **|** void sendeByte**(**const uint8\_t byte**);**

13 **|** //Einzelnes Byte senden

14 **|**

15 **|** void sendeDaten**(**const uint8\_t**\*** data**,** uint16\_t bytes**);**

16 **|** //Daten aus beliebg. vielen Bytes senden

17 **|**

18 **|** void sendeHIGHBit**(**unsigned char anzahl**);**

19 **|** // Ein Manchester HIGH senden (01)

20 **|**

21 **|** void sendeLOWBit**(**unsigned char anzahl**);**

22 **|** // Ein Manchester LOW senden (10)

23 **|**

24 **|** uint16\_t checksumme**(**const uint8\_t**\*** data**,** uint16\_t bytes**);**

25 **|** // Zum Berechnen der Checksumme

26 **|**

27 **|** #endif

# Abbildungsverzeichnis

[Abbildung 1. - Schaltung des Readers (Primär) 7](#_Toc418840103)

[Abbildung 2. - Schaltung des Tags (Sekundär) 7](#_Toc418840104)

[Abbildung 3. - Primär/Sekundär eingespeistes Signal des Funktionsgenerators 8](#_Toc418840105)

[Abbildung 4. - Messpunkte Reader Primäreinspeisung 9](#_Toc418840106)

[Abbildung 5. - Messpunkte Tag Primäreinspeisung 9](#_Toc418840107)

[Abbildung 6. - U1 & U2 (AM Trigger) 10](#_Toc418840108)

[Abbildung 7. - U1 & U2 (Normaler Trigger) 10](#_Toc418840109)

[Abbildung 8. - Spannung an Diode 27 11](#_Toc418840110)

[Abbildung 9. - Eingangssignal zu Ausgangssignal 11](file:///C:\Users\Simon\Dropbox\RF-ID\TILL_4BHELS_Biehl_Hofstaetter_RF-ID_V2.docx#_Toc418840111)

[Abbildung 11. - Messpunkte Reader Sekundäreinspeisung 12](#_Toc418840112)

[Abbildung 12. - Messpunkte Tag Sekundäreinspeisung 12](#_Toc418840113)

[Abbildung 13. - U1 & U2 (AM Trigger) 13](#_Toc418840114)

[Abbildung 14. - U1 & U2 (Normaler Trigger) 13](#_Toc418840115)

[Abbildung 15. - U3 & U4 (Schmitttrigger) 14](#_Toc418840116)

[Abbildung 16. - Eingangssignal zu Ausgangssignal 14](#_Toc418840117)