Dokumentation

für

RF-ID

Klasse	Teammitglied	Signature
4AHELS	Patrik Staudenmayer	
Abgabedatum	Teammitglied	Signature
07.06.2019	Marie Maier	
Lehrer	Teammitglied	Signature
Tillich, Gruber, Crha	-	
Note	Teammitglied	Signature
	-	

Projektbeschreibung

Aufbau und Programmierung eines RF-ID Tags und Lesegerät

USED DEVICES

Nummer	Gerät	Firma	Typ
1	Labornetzteil		
2	Oszilloskope		

Inhaltsverzeichniss

1 Hardware

1.1 Lesegerät

1.1.1 Schaltung

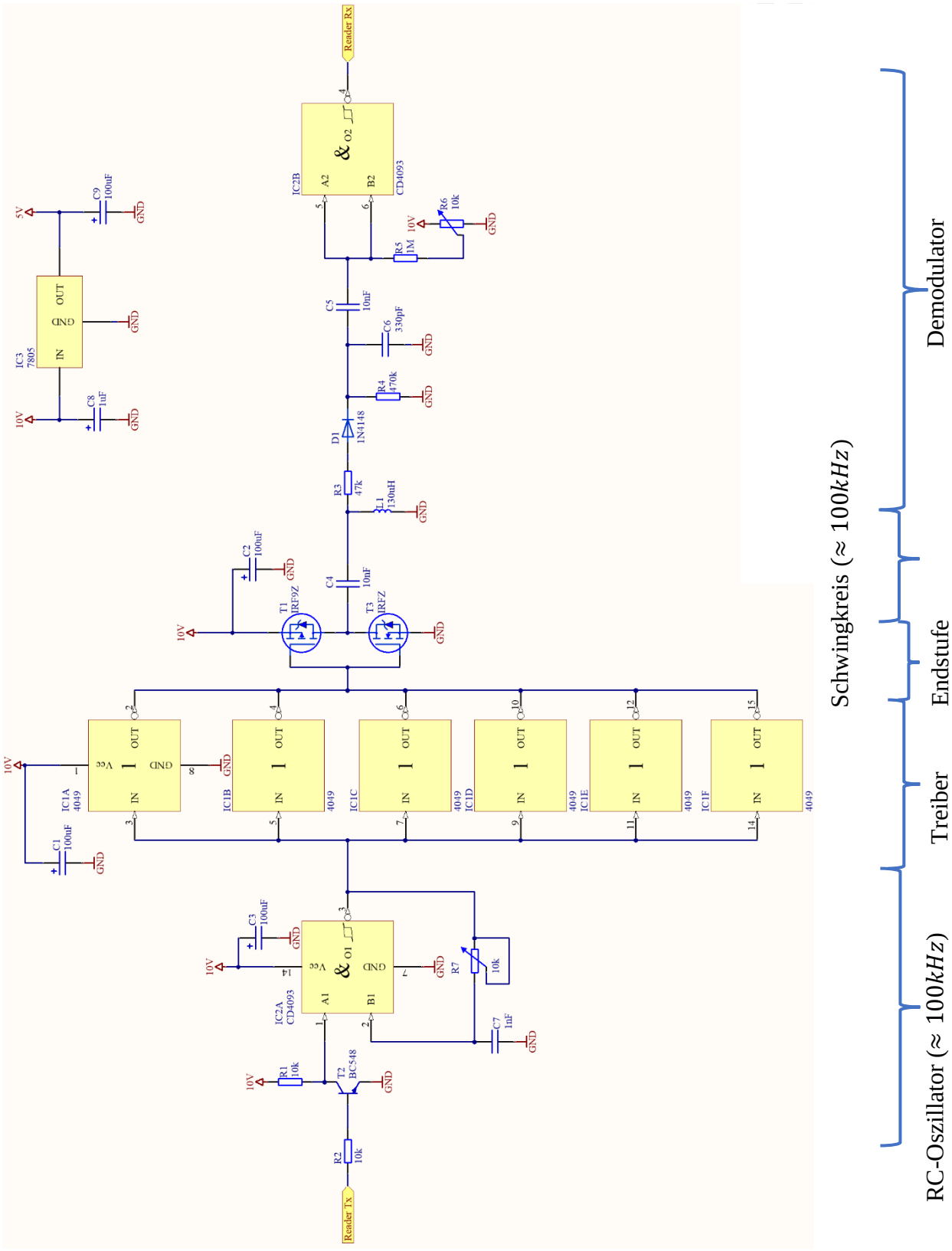


Abbildung 1: Schaltung Lesegerät

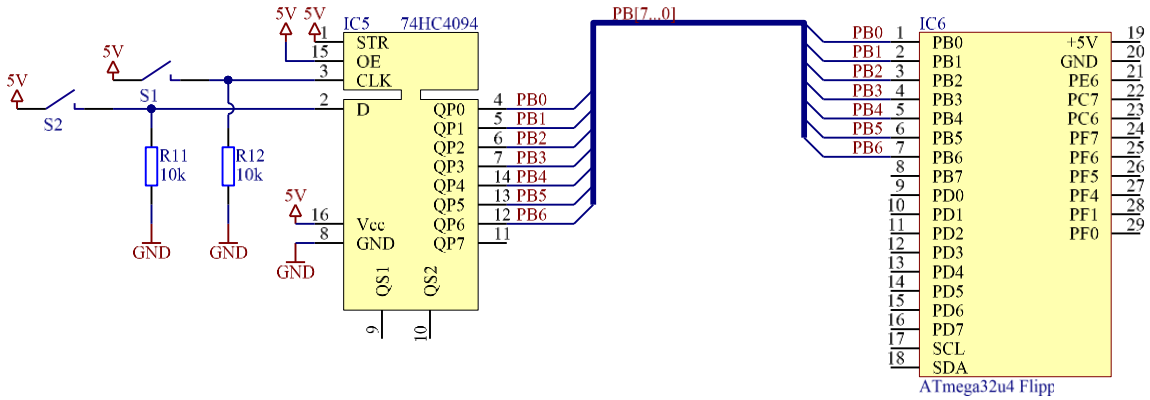


Abbildung 2: Schaltung Lesegerät Daten einlesen

Zum einlesen der Daten wird ein 8 Bit Schieberegister verwendet. Nachdem nur 7 Bits benötigt werden wurden nur sieben der acht Parallelen Ausgänge, des Schieberegister IC5 verwendet.

1.1.3 Berechnungen

1.1.3.1 Resonanzfrequenz RC-Oszillator

$f_0 \approx 100kHz$
$C7 = 1nF$
$f_o \approx \frac{2{,}5}{R7 * C7}$
$\Rightarrow R7 = \frac{2{,}5}{C7 * f_o} = \frac{2{,}5}{1nF * 100kHz} = 25k\Omega$

Durch Testen wurde festgestellt das ein 10k Potentiometer ausreicht, zum einstellen der gewünschten Resonanzfrequenz.

1.1.3.2 Resonanzfrequenz Schwingkreis

$L1 = 130\mu H$
$f_0 \approx 100kHz$
$f_0 = \frac{1}{2 * \pi * \sqrt{L1 * C4}}$
$\Rightarrow C4 = \frac{\left(\frac{1}{f_0 * 2 * \pi}\right)^2}{L1} = \frac{\left(\frac{1}{100kHz * 2 * \pi}\right)^2}{130\mu F} = 19nF$

Ein 10nF Kondensator wurde verwendet, da die Oszillatorfrequenz mit dem Potentiometer abgeglichen werden kann.

1.2 Tag

1.2.1 Schaltung

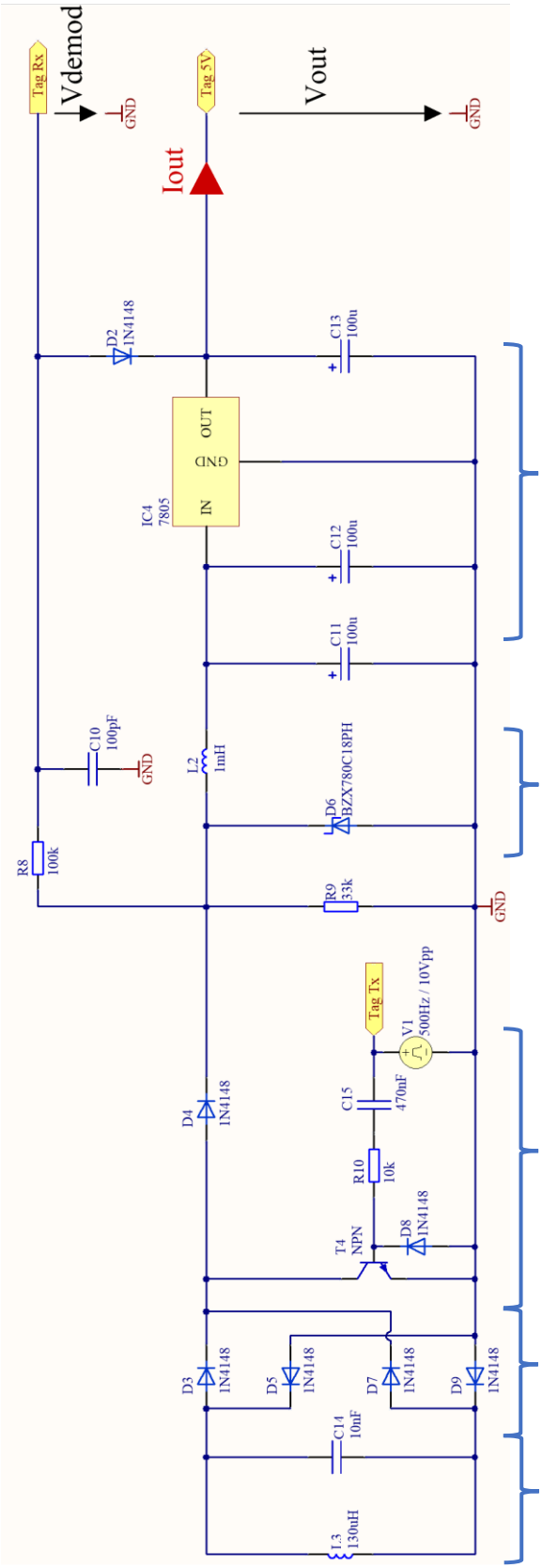


Abbildung 3: Schaltung Tag

1.2.2 Berechnungen

1.2.2.1 Schwingkreis

$L3 = 130\mu H$

$f_0 \approx 100kHz$

$$f_0 = \frac{1}{2 * \pi * \sqrt{L1 * C4}}$$

$$\Rightarrow C14 = \frac{\left(\frac{1}{f_0 * 2 * \pi}\right)^2}{L1} = \frac{\left(\frac{1}{100kHz * 2 * \pi}\right)^2}{130\mu F} = 19nF$$

Ein 10nF Kondensator wurde verwendet, da die Oszillatorfrequenz mit dem Potentiometer am Lesegerät abgeglichen werden kann.

1.2.3 Messungen

1.2.3.1 Übertragungsfunktion der Spannungsversorgung des Tags

1.2.3.1.1 Messschaltung

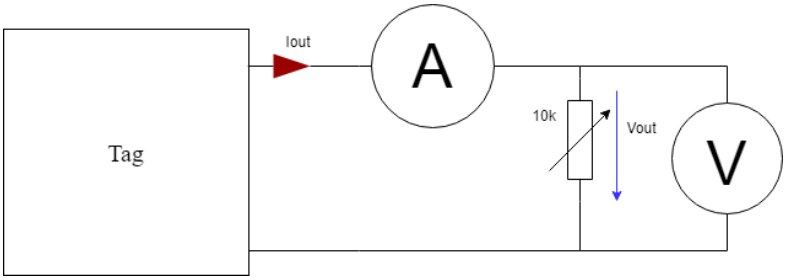
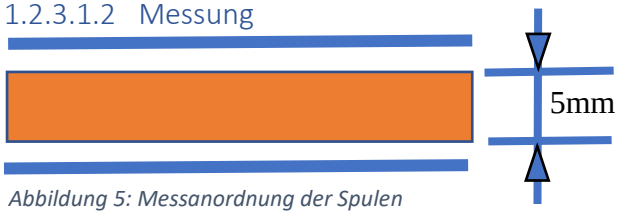


Abbildung 4: Messchaltung für Übertragungsfunktion Spannungsversorgung des Tags

Durch Variierung des 10kΩ Widerstands wurde die Last simuliert.

1.2.3.1.2 Messung



Spule Lesegerät
Plastik Abstandhalter
Spule Tag

Abbildung 5: Messanordnung der Spulen

ssDer ATmega32u4 benötigt 14mA bei einer Taktfrequenz von 16MHz und einer Versorgungsspannung von 5V. Dies kann gerade so abgedeckt werden.

Vou	Iout
5.60 V	0.04 mA
5.60 V	6.63 mA
5.60 V	13.74 mA
5.00 V	18.74 mA
4.60 V	20.67 mA
2.60 V	23.24 mA
0.60 V	26.11 mA

Abbildung 6: Messtabelle Übertragungsfunktion Spannungsversorgung des Tags

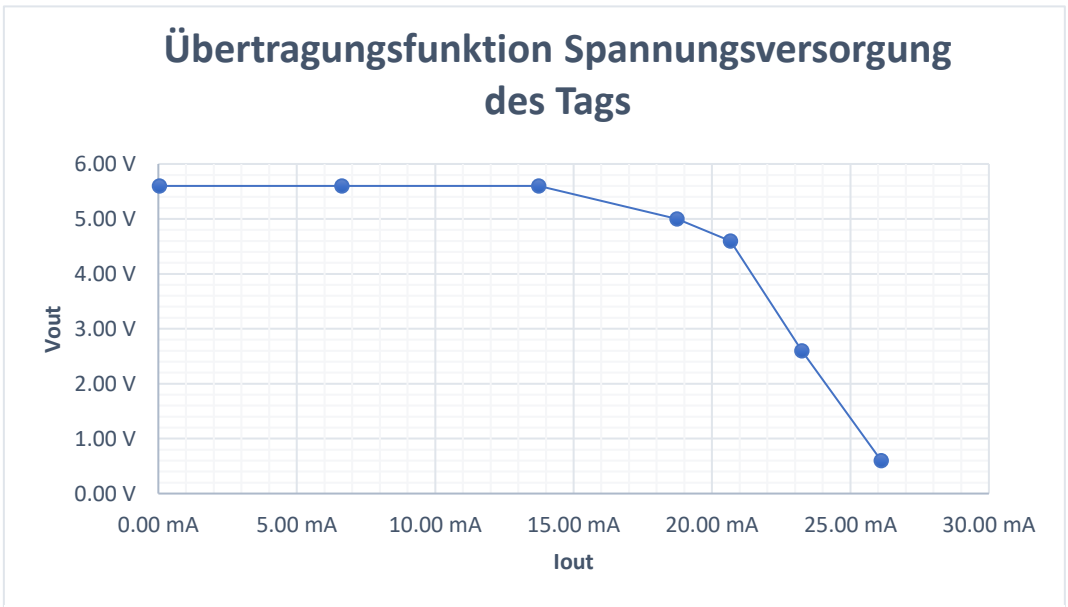


Abbildung 7: Diagramm Übertragungsfunktion Spannungsversorgung des Tags

1.3 Übertragung zwischen Lesegerät und Tag

1.3.1 Übertragung von Tag zu Reader

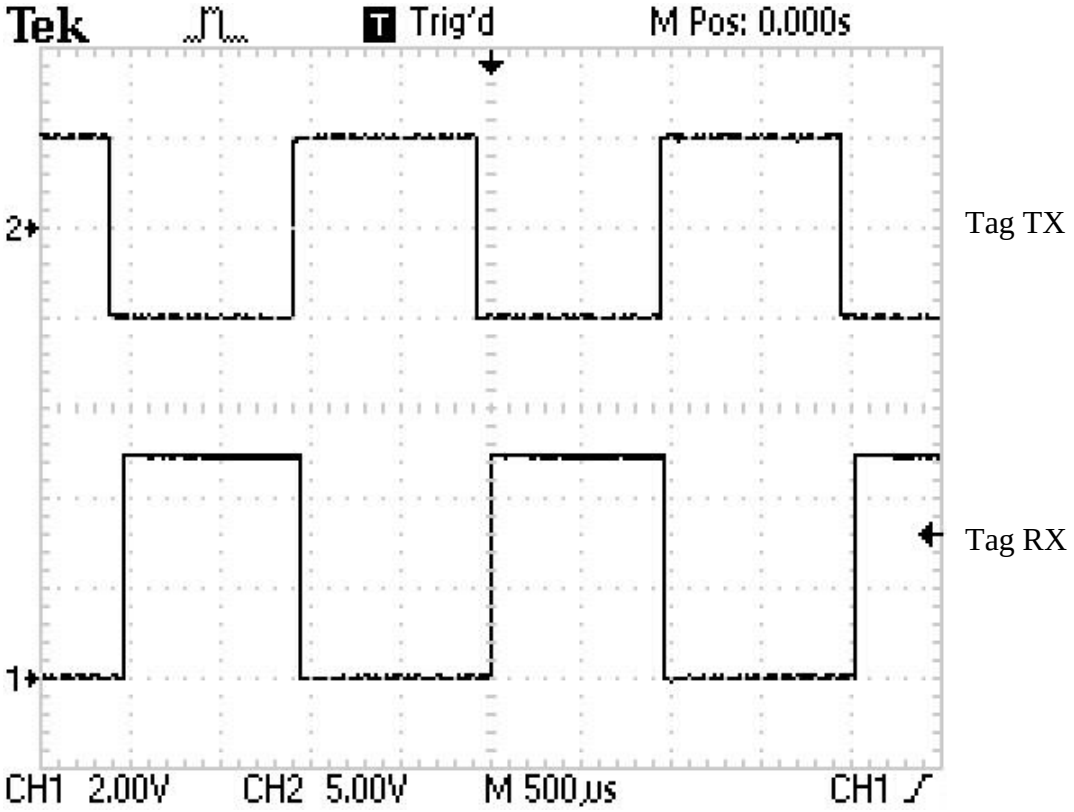


Abbildung 8: Oszillogramm Übertragung von Tag zu Reader

1.3.2 Übertragung von Reader zu Tag

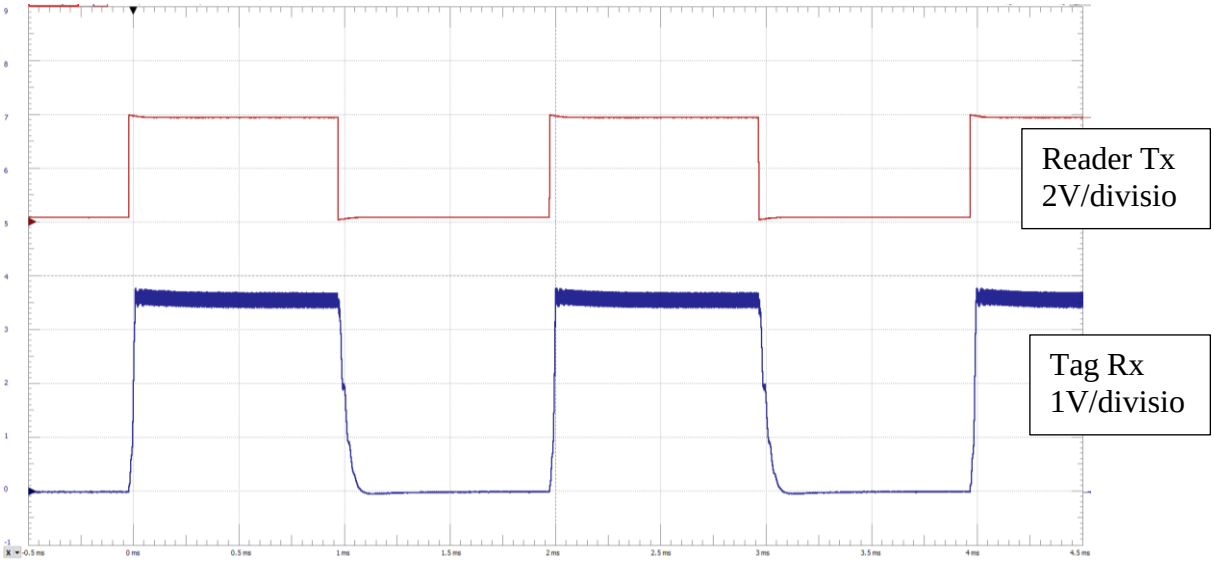


Abbildung 9: Oszillogramm Übertragung von Reader zu Tag

2 Software

Das verwendete Microcontrollerboard ist der „Flipp“ der HTBLuVA St.Pölten Abteilung für Elektronik und technische Informatik, welcher den ATmega32u4 Microcontroller beherbergt.

2.1 Aufgabenstellung

Es soll eine Datenübertragung zwischen zwei Geräten mit folgenden Eigenschaften stattfinden:

- Zum Einlesen der Daten auf einer Seite soll ein SIPO (Seriell In, Parallel Out) Schieberegister verwendet werden.
- Die Daten sollen MSB First gespeichert werden.
- Zum Senden sollen die Daten geschoben werden.
- Die Bitdauer t_{Bit} soll 1,37ms betragen.
- Es soll das MSB zuerst gesendet werden.
- Zur Synchronisierung sollen zwei Startbits gesendet werden.
- Zur Datensicherung soll der Hamming Code verwendet werden.
  Hamming (11/4) … insgesamt 11 Bit und davon 4 Paritätsbits
- Als Leitungscode soll die Manchesterkodierung verwendet werden

2.2 Initialisierung

```
// RX instantiation
char GsaveD[RX_DATA_LENGTH + 1] = {0};
int8_t Gcounter = -4;

// TX instantiation
char GsendData[TX_DATA_LENGTH] = {0};
int8_t GsendCounter = 0;

char RXflag = FALSE;

void RfIDinit()
{
	// Initialize Receiver Port
	RX_PORT_DDR &= ~RX_PIN_MASK;

	// Initial Transmitter Port
	TX_PORT_DDR |= TX_PIN_MASK;
	TX_PORT |= TX_PIN_MASK;                  // Set Tx to high to deliver energy

	// Initialize Input port for shift register
	DATA_INPUT_PORT_DDR = 0x00;

	// Initialize Pin change interrupt
	DDRD &= ~(1 << DDD3);                    // Initialize PD3 as Input
	EICRA |= (1<<ISC31) | (1<<ISC30);        // Trigger interrupt on rising and
	                                            falling edge
	EIMSK |= (1<<INT3);                      // Locally enable Interrupt for INT3


	// Initialize Timer1 for receiving
	TCCR1B = TCCR1B | (1<<WGM13) | (1<<WGM12); // Set Timer1 to CTC-Mode with ICR1
	                                              as TOP
	ICR1 = fICR;                             // Set TOP Value

	OCR1A = fOCR1A;                          // Set threshold for COMP_A
	OCR1B = fOCR1B;                          // Set threshold for COMP_B
	TIMSK0 = TIMSK0 | (1<<OCIE0B) | (1<<OCIE0A) | (1<<ICIE1); //COMP_A, COMP_B  and
	                                                          Input capture as overflow


	// Initialize Timer3 for sending
	TCCR3B = TCCR3B | (1<<WGM33) | (1<<WGM32); // Set Timer1 to CTC-Mode with ICR3
	                                              as TOP
	ICR3 = TOP_TX;                           // Set TOP Value
	TIMSK3 |= (1 << ICIE3);                  // Enable Interrupt on ICIE3

}
```

2.3 Manchester Kodierung

Aus zeitlichen Gründen wurde keine Manchester Kodierung programmiert.

Funktion der Manchester Kodierung:

Wenn Daten Manchester kodiert gesendet werden, werden aus einem Datenbit 2 Datenbits, aus einer 0 wird 10 und aus einer 1 wird 01. Weiters wird der Taktgehalt in der Übertragung erhöht, wodurch eine leichtere Synchronisierung von Sender und Empfänger möglich ist.

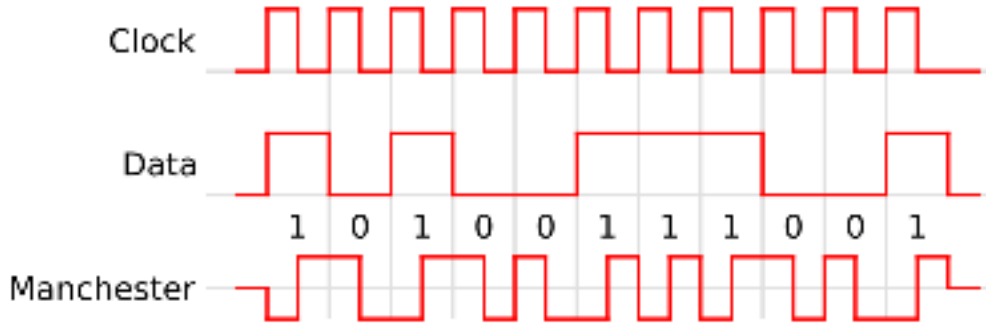


Abbildung 10: Manchester Kodierung

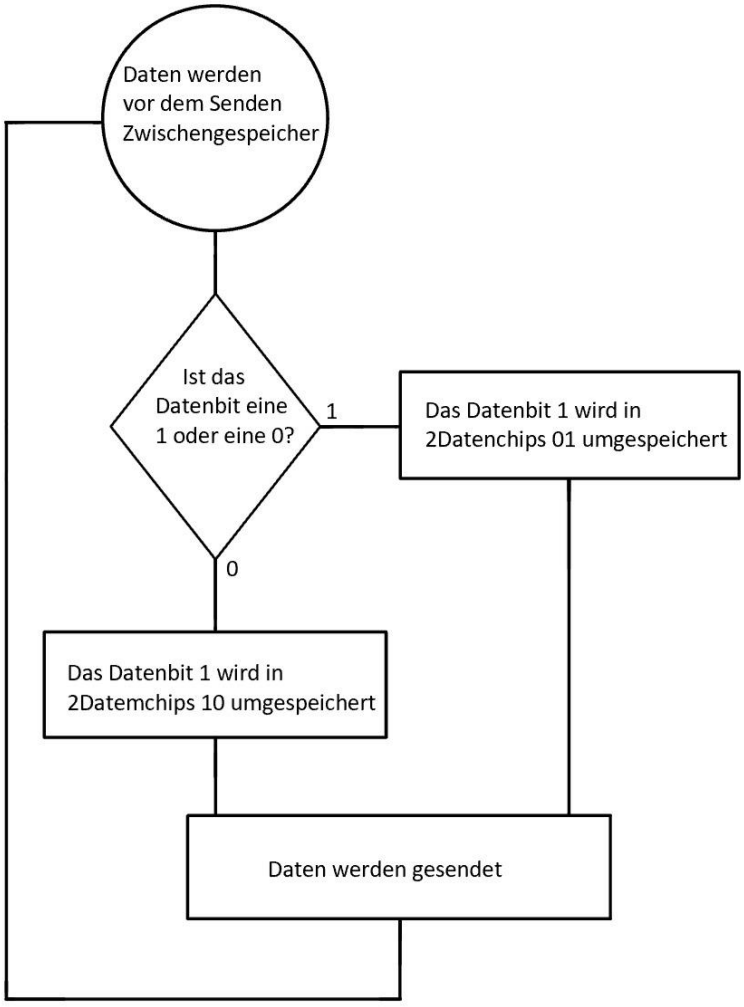


Abbildung 11: Flussdiagramm der Manchester Kodierung

2.4 Hamming Code

Aus zeitlichen Gründen konnte die Hamming Codierung nicht mehr implementiert werden. Der Hamming Code ist ein relativer Einfacher Code mit dem Hamming Abstand 3. Dies bedeutet das zwei Fehler erkannt werden können und ein Fehler korrigiert werden kann.

2.5 Einlesen und Speichern der Daten

Aus zeitlichem Mangel konnte das Einlesen und Speichern der Daten nicht rechtzeitig fertig programmiert werden, daher wurde ein Flussdiagramm erstellt, um einen Überblick über den Ablauf der Einlese und Speicher Routine zu geben.

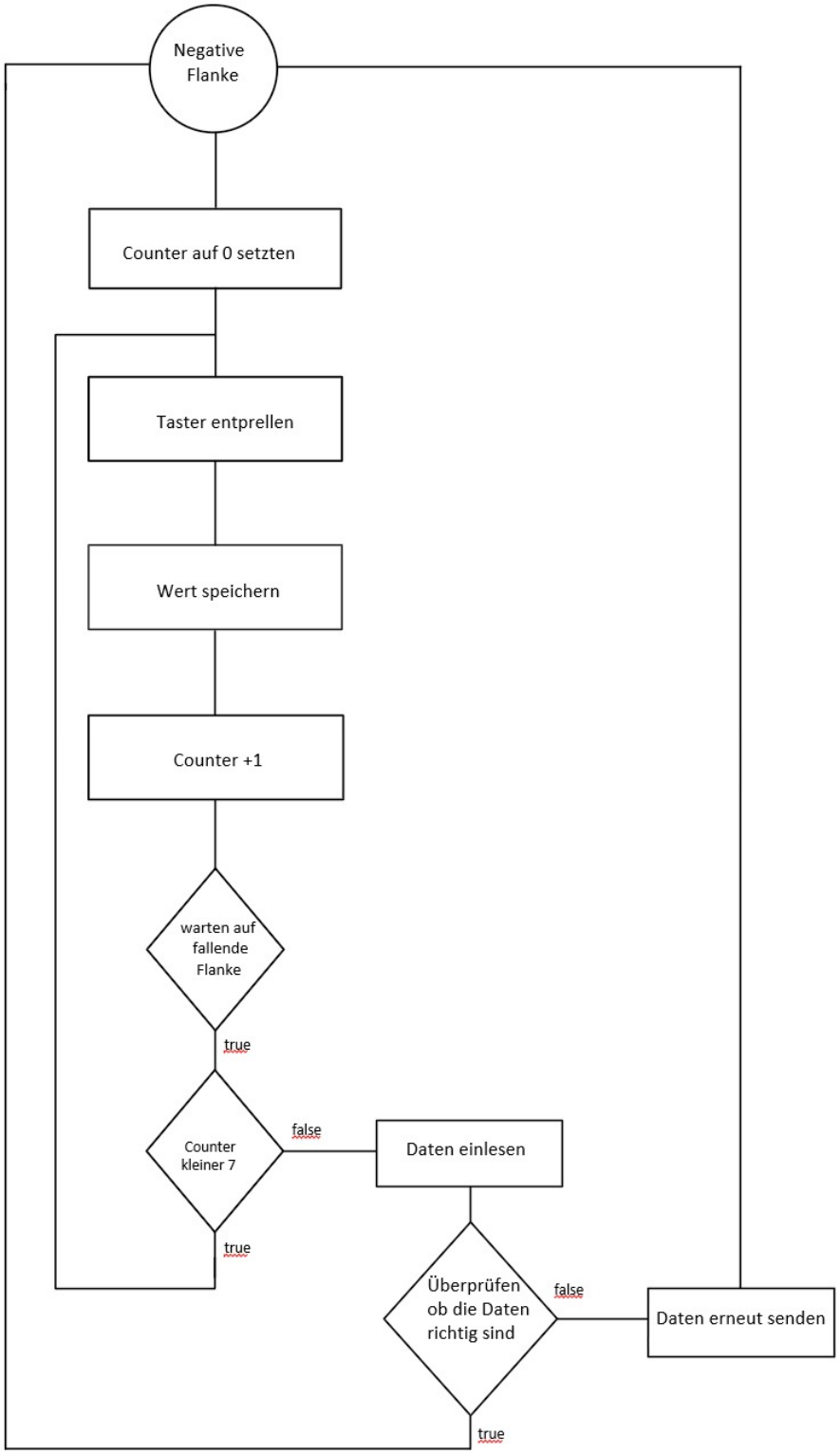


Abbildung 12: Flussdiagramm der Sende- und Einlese-Routine

```
// Input-Counter   PD1(INT1)
#define DATA_LENGTH 7
int InputCounter = 0;
// Initialize Interrupt1 for Input-Counter
DDRD &= ~(1<<DDD1);
EIMSK = EIMSK | (1<<INT1);        //locally enable Interrupt for INT1
EICRA = EICRA | (1<<ISC10) | (1<<ISC11); //set INT1 to generate an Interrupt at the
                                          falling edge
```

Code 2: Initialisierung der Einlese und Speicher Routine

```
//Interrupt Routine
ISR(INT1_vect)
{
	InputCounter++;
	If(InputCounter == 7)
	{
		INPUT_MSB_FIRST(PORTB);
		InputCounter = 0;
	}
}
```

Code 1: Interrupt Routine für Einlesen und Speichern

```
#define INPUT_MSB_FIRST(input)  ({                          \
	unsigned char ret[DATA_LENGTH];                         \
	char tmp = input;                                       \
	do {                                                    \
		for (int i = 0; i < DATA_LENGTH; i++)               \
		{                                                   \
		   ret[DATA_LENGTH - 1 - i] = (((1 << i) & tmp) == (1 << i) ? 1: 0);\
		}                                                                    \
	} while(0);                                             \
	ret;                                                    \
	})
```

Code 3: Speichern der Daten

2.6 Senderoutine

2.6.1 Senderoutine Starten

Um die Senderoutine zu starten wird das Makro RF_ID_SEND (Code 1) verwendet. Diese führt die in der Abbildung 14: Flussdiagram des Sende Makros dargestellten Tätigkeiten durch. Hierbei ist zur Implementierung anzumerken, dass, wenn die Übergabe der Daten mittels Globaler Variablen implementiert wird, welche sowohl im Main als auch in der Library verwendbar sein sollen, so muss das C Schlüsselwort extern verwendet werden.
Variablen, die mit dem Schlüsselwort extern deklariert werden, werden bei der Deklaration noch kein Speicher zugewiesen, dies geschieht erst bei der Definition welche bei der hier gewählten Implementation in dem „RF_ID.c“ File stattfindet.
Um die Synchronität zu gewährleisten wird am Anfang einer jeden Transaktion die Bits „11“ Manchester kodiert gesendet.

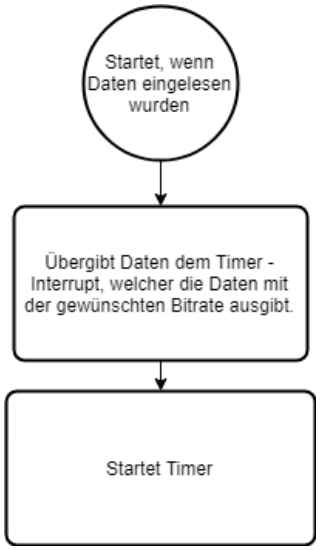


Abbildung 13: Flussdiagram des Sende Makros

```
/*
 *@function                             RfIDsend
 *@abstract                             Handles the transmission of the given data
 *@discussion                           Calculates the Hamming code of the given and
                                        encodes it to Manchester code then handles
                                        the correct set up of the timer
 *                                      and assigns the encoded data to GsendData.
 *@param            data                Is the pointer to the transmitted data
 */
#define RF_ID_SEND(data) ({                                                     \
       for(int i = 0; i < TX_DATA_LENGTH; i++)                                  \
       {                                                                        \
              GsendData[i] = data[i];                                           \
       }                                                                        \
       GsendCounter =       0;                                                  \
                                                                                \
       TCCR3B |= (1 << CS30);           /* Start Timer3 (Set divider to 1) */   \
       })
```

Code 4: Makro zum Starten der Senderoutine

2.6.2 Timer der Senderoutine

Um eine fixe Bitrate zu gewährleisten wurde der Timer 3 des ATmega32u4 im CTC Modus verwendet. Ein Bit wird gesendet, wenn dieser Timer den Topwert erreicht hat, welcher mittels des ICR3 Registers festgelegt wurde und welcher beim Erreichen den Timer 3 Capture Interrupt auslöst, wenn dieser freigeschalten ist.

2.6.2.1 Berechnung der Timer Parameter

Anforderung: Es soll alle 1,37ms ein Bit gesendet werden.

$$f_{OCnA} = \frac{f_{CLK\_I/O}}{2 * N * (1 + OCRnA)}$$ [1]

$$\Rightarrow \frac{1}{t_{Bit}} = \frac{f_{CLK\_I/O}}{2 * N * (1 + OCRnA)}$$

$$\Rightarrow OCRnA = \frac{f_{CLK\_I/O} * t_{Bit}}{2 * N} - 1$$

$N \in \{1; 8; 64; 256; 1024\}$[2]

Der Microcontroller wird mit einer $f_{CLK\_I/O}$ von 16MHz betrieben.

Mithilfe von Excel wurde folgende Tabelle erstellt:

Teiler N	ICR3	
1	10959.00	← gewählt
8	1369.00	
64	170.25	
256	41.81	
1024	9.70	

Abbildung 14: Mögliche Werte für den Timer der Senderoutine

Die Kriterien, die hier erfüllt werden mussten, war:

- $< 2^{16}$, da Timer 3 ein 16 Bit Timer ist.
- > 200, um falls nötig die Grenze noch verstellen zu können.

[1] Entnommen aus dem Datenblatt des ATmega32u4 im Kapitel 14.8.2 http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf am 28.05.2019

[2] Siehe Fußnote 1

2.6.2.2 *Interruptroutine des Timer 3 Capture Interrupt*

In Abbildung 16: Flussdiagram Interrupt Routine Timer 3 Capture Interrupt ist dargestellt wie das senden in der Interrupt Routine bewerkstelligt wird.

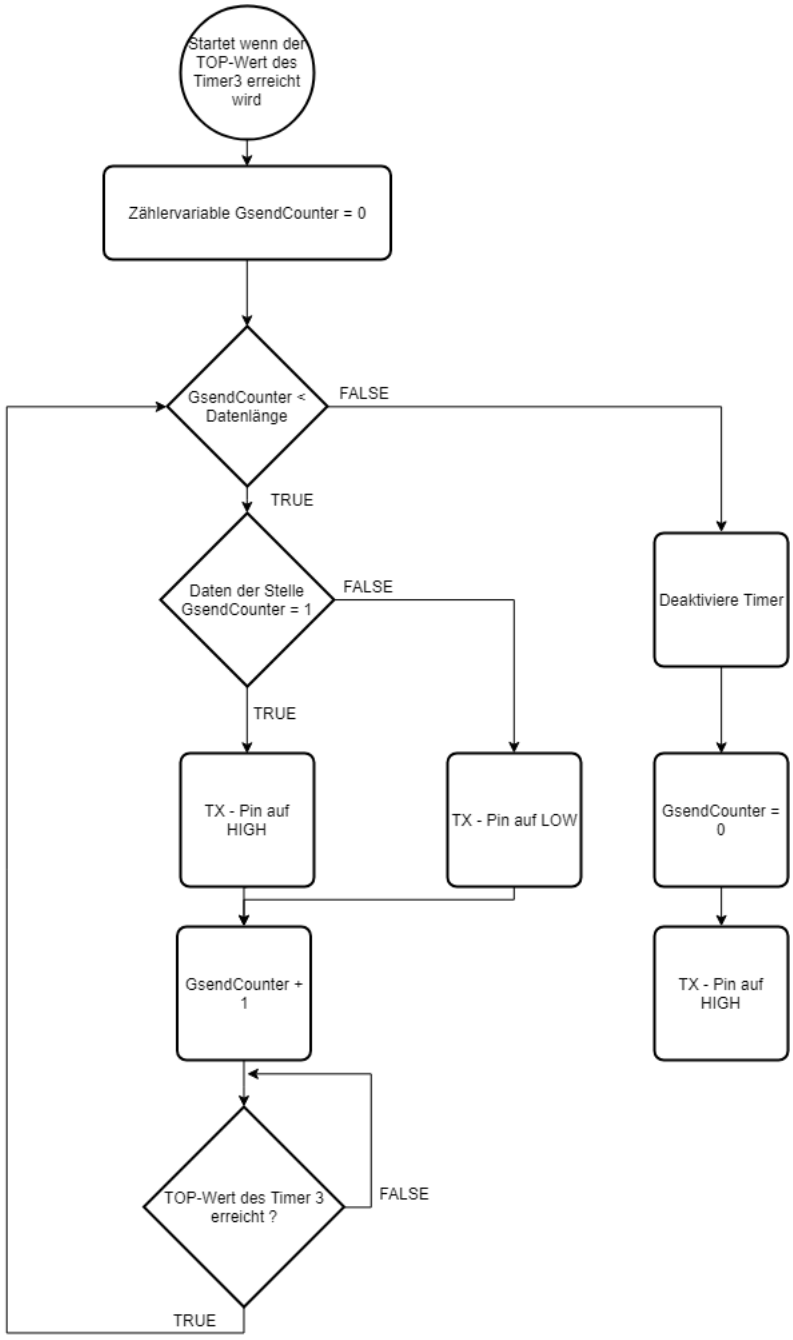


Abbildung 15: Flussdiagram Interrupt Routine Timer 3 Capture Interrupt

```
ISR(TIMER3_CAPT_vect)
{
	// Send Interrupt
	if (GsendCounter < TX_DATA_LENGTH)
	{
		if(GsendData[GsendCounter] == 1)
		{
			TX_PORT |= TX_PIN_MASK;
		}
		else
		{
			TX_PORT &= ~TX_PIN_MASK;
		}

		GsendCounter++;
	}
	else
	{
		GsendCounter = 0;
		TCCR3B &= ~(1 << CS32) & ~(1 << CS31) & ~(1 << CS30);
		// Deactivate Timer3
		TCNT3 = 0;		// Reset Timer3
		TX_PORT |= TX_PIN_MASK;
	}
}
```

Code 5: Sende Interrupt Routine des Timer 3 Capture Interrupt

2.7 Empfangsroutine

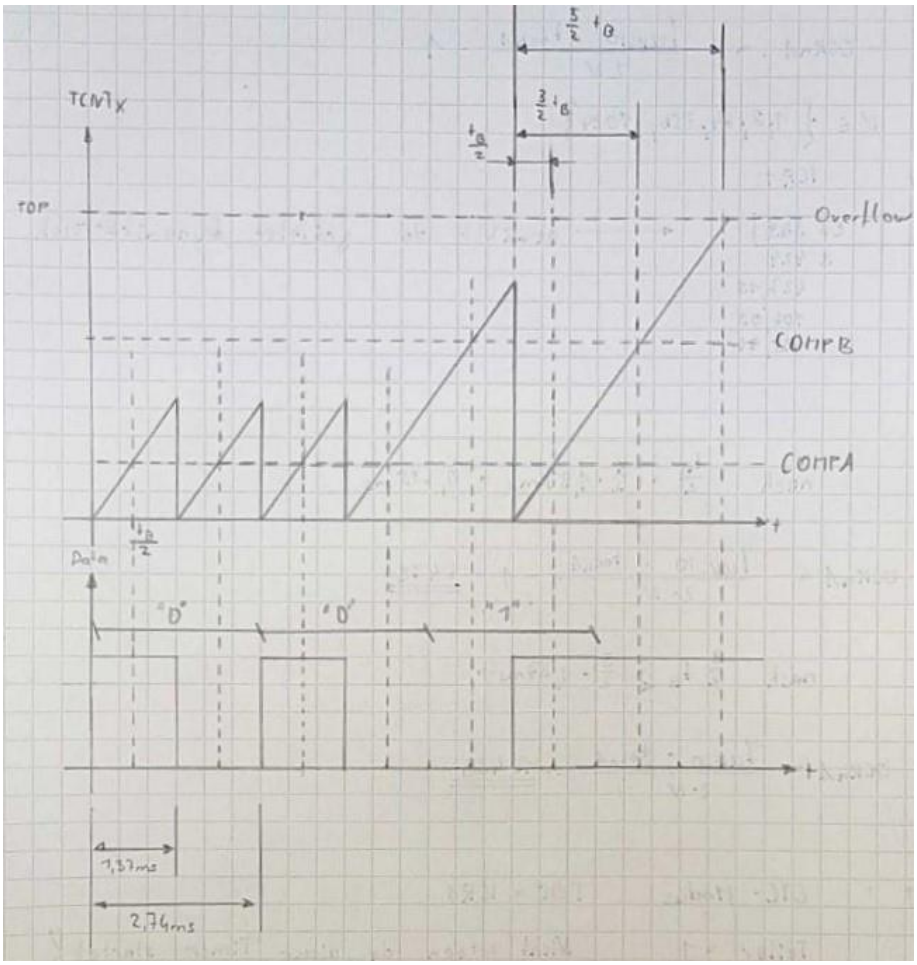


Abbildung 16: Timing Diagramm der Empfangsroutine

Zum Empfangen, der gesendeten Daten, wird der 16-Bit Timer 1 im CTC-Modus verwendet. Der COMPA Interrupt wird verwendet, um das erste Bit des Symbols zu empfangen, der COMPB Interrupt wird verwendet, um das erste Bit des darauffolgenden Symbols zu empfangen, falls die beiden verschiedene Symbole, des Manchester Codes, hintereinander gesendet wurden. Der Overflow Interrupt dient zum Erkennen des Endes der Datenübertragung.
Der Pin-Change Interrupt INT3 – PD3 des ATmega32u4 – wird verwendet, um den Sender und Empfänger Synchron zu halten. Im Interrupt wird bei einer Flanke das Counter Register des Timer 1 auf 0 gesetzt und dadurch ist die Synchronität des Sender und Empfängers gewährleistet.

2.7.1 Timer der Empfangsroutine

2.7.1.1 Berechnung der Timer Parameter

Anforderung:

- COMPA soll nach 685ns erreicht werden.
- COMPB soll nach 2,055ms erreicht werden.
- TOP soll nach 3,425ms erreicht werden.

$$f_{OCnA} = \frac{f_{CLK\_I/O}}{2 * N * (1 + OCRnA)}$$ [3]

$$\Rightarrow \frac{1}{t_{Bit}} = \frac{f_{CLK\_I/O}}{2 * N * (1 + OCRnA)}$$

$$\Rightarrow OCRnA = \frac{f_{CLK\_I/O} * t_{Bit}}{2 * N} - 1$$

$N \in \{1; 8; 64; 256; 1024\}$[4]

Der Microcontroller wird mit einer $f_{CLK\_I/O}$ von 16MHz betrieben.

Mithilfe von Excel wurden folgende Tabellen erstellt:

Divider N	OCRnA (Register zum Festlegen der Grenzte für den COMPA Interrupt)	
1	5479.00	gewählt
8	684.00	
64	84.63	
256	20.41	
1024	4.35	

Abbildung 17: Mögliche Werte für den Wert des OCRnA Registers

Divider N	OCRnB (Register zum Festlegen der Grenze für den COMPB Interrupt)	
1	16439.00	gewählt
8	2054.00	
64	255.88	
256	63.22	
1024	15.05	

Abbildung 18: Mögliche Werte für den Wert des OCRnB Registers

Divider N	ICRn (Register zum Festlegen der Grenze für den Input Capture Interrupt)	
1	27399.00	gewählt
8	3424.00	
64	427.13	
256	106.03	
1024	25.76	

Abbildung 19: Mögliche Werte für den Wert des ICRn Registers

[3] Entnommen aus dem Datenblatt des ATmega32u4 im Kapitel 14.8.2 http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf am 28.05.2019

[4] Siehe Fußnote 1

2.7.2 Flussdiagram der Empfangsroutine

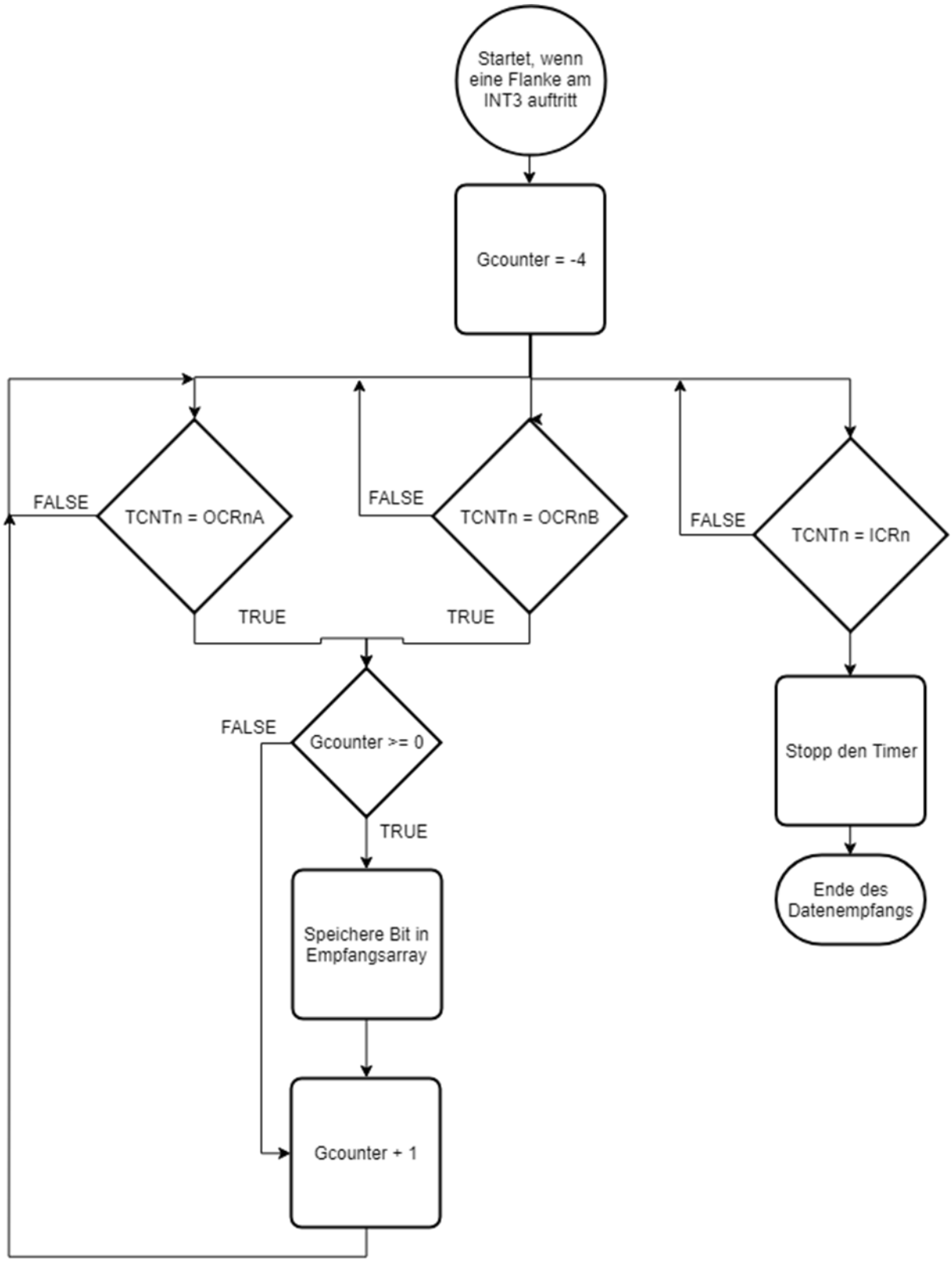


Abbildung 20: Flussdiagram der Empfangsroutine

2.7.3 Code der Empfangsroutine

```
ISR(TIMER1_COMPA_vect)
{
	// First routine to be triggered when something received
	if (Gcounter >= 0)
	{
		GsaveD[Gcounter] = RX_PORT & RX_PIN_MASK;
	}

	Gcounter++;
}


ISR(TIMER1_COMPB_vect)
{
	// Second routine to be triggered when something received

	if (Gcounter >= 0)
	{
		GsaveD[Gcounter] = RX_PORT & RX_PIN_MASK;
	}

	Gcounter++;
}


ISR(TIMER1_CAPT_vect)
{
	// Ends the transaction

	Gcounter = -4;
	RXflag = TRUE;

	TCCR1B = TCCR1B & ~(1<<CS10) & ~(1<<CS11) & ~(1<<CS12);	//Timer1 stopped
}
```

Code 6: Implementierung der Empfangsroutine