

# **HN4-FTKL**

## **Abteilung Elektronik**

an der Höheren technischen Bundeslehranstalt 1  
Innsbruck, Anichstraße 26 – 28

# **AquadDoc**

**Manuel Ljubic, Jack Neuner, Daniel Plank**

**2019-06-23**

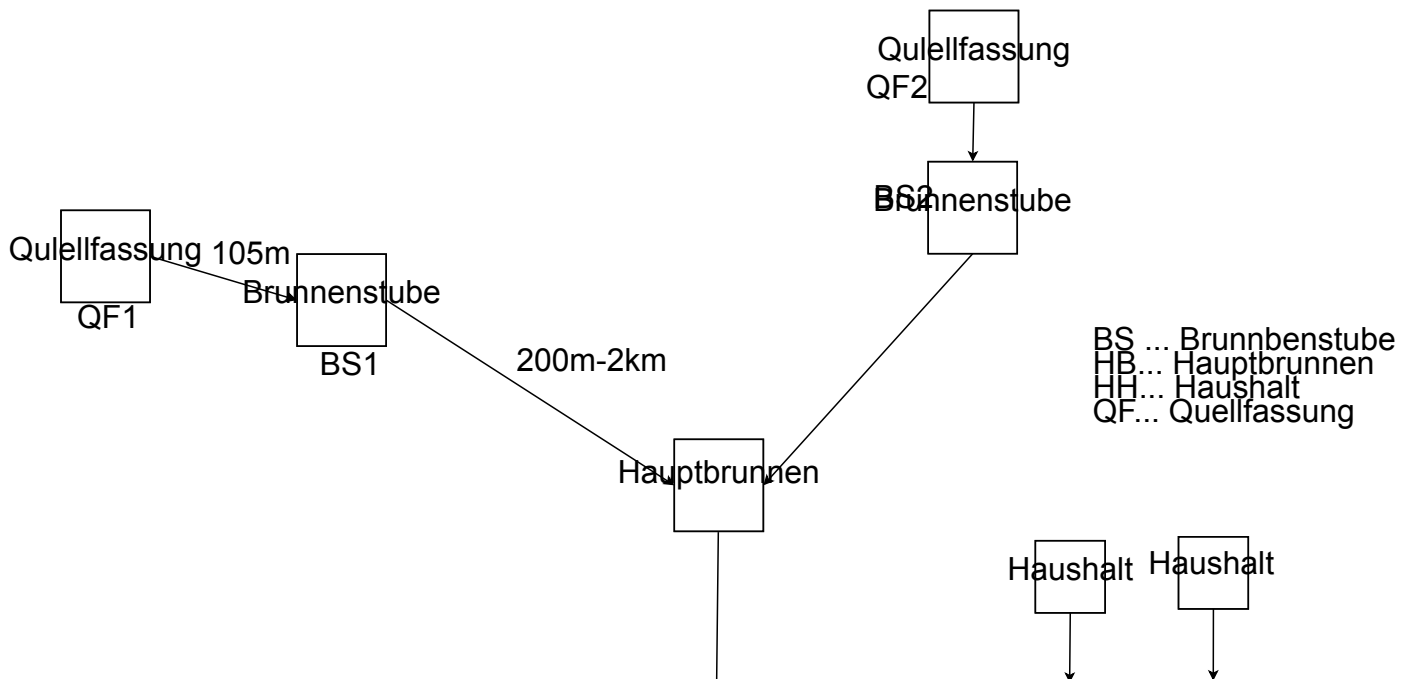
**Dokumentation des Wasserhaushalts einer Wasserversorgungsanlage für  
Kleinsiedlungsgebiete**

# Inhaltsverzeichnis

<b>1 Aufgabenstellung</b>	<b>2</b>
1.1 Aufbau . . . . .	2
1.2 Sensoren . . . . .	2
<b>2 Aufgabenverteilung</b>	<b>3</b>
2.1 Ljubic . . . . .	3
2.2 Neuner . . . . .	4
2.3 Plank . . . . .	4
<b>3 Test mittels Arduino</b>	<b>4</b>
3.1 HC-SR04 . . . . .	6
3.2 BMP180 . . . . .	7
<b>4 Durchführung Mittels PSoC</b>	<b>10</b>
4.1 Timer-Interrupt . . . . .	10
4.1.1 C-Code . . . . .	10
4.1.2 Top-Sheet . . . . .	11
4.2 Spannungsversorgung . . . . .	11
4.3 Bauteilerklärung . . . . .	11
4.3.1 C-Code . . . . .	11
4.3.2 Top-Sheet . . . . .	13
4.4 HC-SR04 . . . . .	13
4.4.1 Bauteilerklärung . . . . .	13
4.4.2 C-Code . . . . .	14
4.4.3 Top-Sheet . . . . .	16
4.5 Durchflussmessung . . . . .	16
4.5.1 Bauteilerklärung . . . . .	16
4.5.2 Prellen des REED-Kontaktes . . . . .	17
4.5.3 C-Code . . . . .	18
4.5.4 Top-Sheet . . . . .	19
4.6 DS1722 . . . . .	20
4.6.1 Bauteilerklärung . . . . .	20
4.6.2 C-Code für DS1722 . . . . .	21
4.6.3 Top-Sheet . . . . .	22
4.7 BIM2 . . . . .	22
4.7.1 Bauteilerklärung . . . . .	22
4.7.2 Anschluss . . . . .	23
4.7.3 C-Code für BIM2 . . . . .	23
4.7.4 Übertragungsbeispiele . . . . .	24
<b>5 Anhang</b>	<b>25</b>
5.1 Literaturverzeichnis . . . . .	25

# 1 Aufgabenstellung

## 1.1 Aufbau



Ein Hochbehälter hat ein Fassungsvermögen um 1 Tag Wasser speichern zu können oder ein Feuer zu löschen.

## 1.2 Sensoren

- Wasser zwischen Quelle und Brunnen und Brunnen und Häuser  
Umsetzung mittels Rotor mit Magnet an einer Schaufel, welcher einen REED-Kontakt schaltet. Aufgabe: Prellt dieser Schalter?

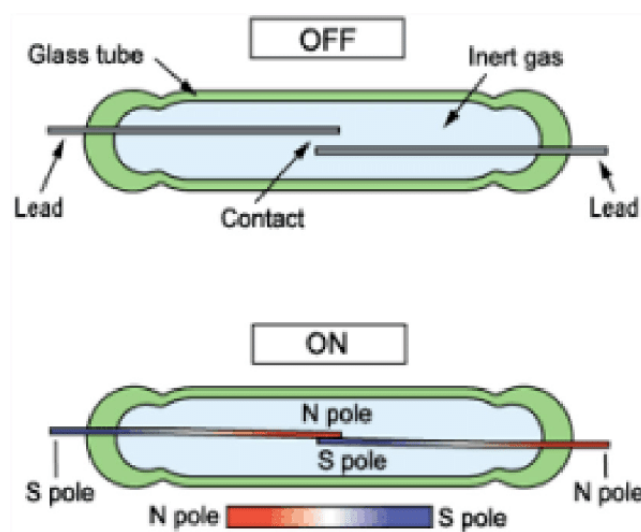


Abbildung 1: Schemata eines Reed-Schalters

- Wassertemperatur  
Digitaler Sensor → Auflösung von 0.01°C

- Wasserstand Füllstand HB: Drucksensor, Ultraschallsensor, Potentiometer mit Schwimmer, etc.
- Türschalter
- Batteriestandsanzeige

Anforderungen an die Funkübertragung:

- Datensicherheit
- Übertragungsmöglichkeit (Manchester Kodierung)
- Übertragungsantenne

Gefundene Bauteile auf neuhold-elektronik.at:

- DS1722:  
SPI Digital Thermometer. 8-12 bit Auflösung.
- HC - SR04:  
PWM Ultraschall Messmodul. 2mA standby strom.
- MAX640:  
5V Step-Down DC-DC Converter.

## 2 Aufgabenverteilung

- Plank: Programmentwicklung, testen der Sensoren und Teile der Dokumentation
- Neuner: Programmentwicklung und erstellen des Top Design
- Ljubic: Dokumentation

### 2.1 Ljubic

Datum	Beschreibung	Stunden
04.02.19	Aufgabenbesprechung	3h
11.02.19	Entfall	0h
18.02.19	Erstellen des Struktogramm	3h
25.02.19	HC-SR04 Ultraschallsensor programmieren	3h
04.03.19	Dokumentation + Fertigstellen des Struktogramm	3h
11.03.19	Dokumentation	3h
18.03.19	Entfall	0h
25.03.19	Dokumentation + DS1722 Temperatursensor programmieren	3h
01.04.19	Dokumentation	3h
08.04.19	Dokumentation	3h
29.04.19	Dokumentation + Nachholen des Zeitplans	3h
06.05.19	Dokumentation + Einfügen der C-Codes in Dokumentation	3h
13.05.19	Dokumentation + Abgabenbesprechung	2h
27.05.19	Dokumentation	3h
03.06.19	Dokumentation + Verbindung mit Spektrumanalyser getestet	3h

## 2.2 Neuner

Datum	Beschreibung	Stunden
04.02.19	Aufgabenbesprechung	3h
11.02.19	Entfall	0h
18.02.19	Sensoren ausgewählt	3h
25.02.19	HC-SR04 Ultraschallsensor programmieren	3h
04.03.19	HC-SR04 Ultraschallsensor Verfeinerung	3h
11.03.19	DS1722 Temperatursensor programmieren	3h
18.03.19	Entfall	0h
25.03.19	Batteriestandmessung + DS1722 Temperatursensor programmieren	3h
01.04.19	DS1722 Temperatursensor programmieren	3h
08.04.19	DS1722 Temperatursensor programmieren	3h
29.04.19	BMP180 Temperatursensor programmieren	3h
06.05.19	BMP180 Temperatursensor programmieren	3h
13.05.19	BMP180 Temperatursensor programmieren + Abgabenbesprechung	2h
27.05.19	BMP180 Temperatursensor programmieren	3h
03.06.19	PSOC-Programm für Empfangen der Daten + Verbindung mit Spektrumanalyse getestet	3h

## 2.3 Plank

Datum	Beschreibung	Stunden
04.02.19	Aufgabenbesprechung	3h
11.02.19	Frei	0h
18.02.19	Sensoren ausgewählt + Beginn der Dokumentation	3h
25.02.19	HC-SR04 Ultraschallsensor programmieren + Dokumentationsweiterung	3h
04.03.19	HC-SR04 Ultraschallsensor Verfeinerung	3h
11.03.19	DS1722 Temperatursensor programmieren	3h
18.03.19	Entfall	0h
25.03.19	DS1722 Temperatursensor programmieren	3h
01.04.19	DS1722 Temperatursensor programmieren	3h
08.04.19	DS1722 Temperatursensor programmieren	3h
29.04.19	BMP180 Temperatursensor programmieren	3h
06.05.19	DS1722 Temperatursensor programmieren	3h
13.05.19	BMP180 Temperatursensor programmieren + Abgabenbesprechung	2h
27.05.19	BMP180 Temperatursensor programmieren	3h
03.06.19	Verbindung mit Spektrumanalyse getestet	3h

## 3 Test mittels Arduino

Es wird ein Arduino verwendet um die gegebenen Sensoren DS1722 bzw. HC-SR04 zu testen. Die SPI-Anschlüsse des Arduinos werden mit den Anschlüssen der jeweiligen Sen-

soren verbunden ebenfalls wird auf die Verwendung der richtigen Versorgungsspannung geachtet.

### 3.1 HC-SR04

Listing 1: Arduino Programm für HC-SR04

```
1  /*
2  * Ultrasonic Sensor HC–SR04 and Arduino Tutorial
3  *
4  * by Dejan Nedelkovski,
5  * www.HowToMechatronics.com
6  *
7  */
8
9  // defines pins numbers
10 const int trigPin = 9;
11 const int echoPin = 10;
12
13 // defines variables
14 long duration;
15 int distance;
16
17 void setup() {
18     pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
19     pinMode(echoPin, INPUT); // Sets the echoPin as an Input
20     Serial.begin(9600); // Starts the serial communication
21     return;
22 }
23
24 void loop() {
25     // Clears the trigPin
26     digitalWrite ( trigPin , LOW);
27     delayMicroseconds(2);
28
29     // Sets the trigPin on HIGH state for 10 micro seconds
30     digitalWrite ( trigPin , HIGH);
31     delayMicroseconds(10);
32     digitalWrite ( trigPin , LOW);
33
34     // Reads the echoPin, returns the sound wave travel time
35     // in µS
36     duration = pulseIn(echoPin, HIGH);
37
38     // Calculating the distance
39     distance= duration*0.034/2;
40
41     // Prints the distance on the Serial Monitor
42     Serial.print("Distance: ");
43     Serial.println (distance);
44     return;
45 }
```

## 3.2 BMP180

Listing 2: Arduino Programm für DS1722

```

1  #define DATAOUT 11 //MOSI – Master Input Slave Output
2  #define DATAIN 12 //MISO – Master Output Slave Input
3  #define SPICLOCK 13 //SCK – Serial Clock
4  #define SLAVESELECT 10 //SS – Slave Select
5  #define DS1722_POWER 9
6
7  #define DS1722_SELECT HIGH
8  #define DS1722_DESELECT LOW
9
10 #define DS1722_CONFIG_BYTE 0xEE
11 #define CONFIG_REG_READ 0x00
12 #define CONFIG_REG_WRITE 0x80
13 #define TEMP_ADDR_HI 0x02
14 #define TEMP_ADDR_LOW 0x01
15
16 byte clr;
17 byte temperature[2];
18
19 char spi_transfer(volatile char data){
20
21     // Start the transmission
22     SPDR = data;// Wait the end of the transmission
23     while (!(SPSR & (1<<SPIF))){};
24     // return the received byte
25     return SPDR;
26 }
27
28 void setup(){
29
30     byte n, config = 0xAB;
31     Serial.begin(9600);
32
33     temperature[0] = 0x12;
34     temperature[1] = 0x34;
35
36     /* Set DDIR registers */
37
38     pinMode(DATAOUT, OUTPUT);
39     pinMode(DATAIN, INPUT);
40     pinMode(SPICLOCK, OUTPUT);
41     pinMode(SLAVESELECT, OUTPUT);
42     pinMode(DS1722_POWER, OUTPUT);
43
44     digitalWrite (DS1722_POWER, HIGH); //disable device
45     delay(250);
46

```



```
47     digitalWrite (SLAVESELECT, DS1722_DESELECT); //disable device
48     // set up SPI control register
49     SPCR = (1<<SPE)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA);
50     clr=SPSR;
51     clr=SPDR;
52     delay(10);
53
54     // read config byte
55     digitalWrite (SLAVESELECT, DS1722_SELECT);
56     spi_transfer(CONFIG_REG_READ);
57     config = spi_transfer(0xFF);
58     digitalWrite (SLAVESELECT, DS1722_DESELECT);
59     delay(100);
60
61     Serial.print(config, HEX);
62     Serial.print('\n', BYTE);
63
64     // write config byte to the configuration register
65     digitalWrite (SLAVESELECT, DS1722_SELECT);
66     spi_transfer(CONFIG_REG_WRITE);
67     spi_transfer(DS1722_CONFIG_BYTE);
68     digitalWrite (SLAVESELECT, DS1722_DESELECT);
69     delay(100);
70
71     // read config byte
72     digitalWrite (SLAVESELECT, DS1722_SELECT);
73     spi_transfer(CONFIG_REG_READ);
74     config = spi_transfer(0xFF);
75     digitalWrite (SLAVESELECT, DS1722_DESELECT);
76     delay(100);
77
78     Serial.print(config, HEX);
79     Serial.print('\n', BYTE);
80
81     Serial.print('h', BYTE);
82     Serial.print('i', BYTE);
83     Serial.print('\n', BYTE); //debug
84     delay(1000);
85
86     return;
87 }
88
89 void loop(){
90
91     // float c, f;
92     // write config byte to the configuration register
93     digitalWrite (SLAVESELECT, DS1722_SELECT);
94     spi_transfer(CONFIG_REG_WRITE);
95     spi_transfer(DS1722_CONFIG_BYTE);
96     digitalWrite (SLAVESELECT, DS1722_DESELECT);
```

```
97
98     delay(1400);
99
100     digitalWrite (SLAVESELECT,DS1722_SELECT);
101     spi_transfer (TEMP_ADDR_HI);
102     temperature[0] = spi_transfer(0x00);
103     //release chip, signal end transfer
104     digitalWrite (SLAVESELECT, DS1722_DESELECT);
105     delay(25); // just because....
106     digitalWrite (SLAVESELECT,DS1722_SELECT);
107     spi_transfer (TEMP_ADDR_LOW);
108     temperature[1] = spi_transfer(0x00);
109     //release chip, signal end transfer
110     digitalWrite (SLAVESELECT, DS1722_DESELECT);
111     Serial . print (temperature[0] * 9 / 5 + 32, DEC);
112     Serial . print ( ' ', BYTE);
113     Serial . print (temperature[0], DEC);
114
115     if (temperature[1] & 0x80){
116
117         Serial . print ( ' ', BYTE);
118         Serial . print ( '5', BYTE);
119     }
120
121     Serial . print ( '\r', BYTE);
122     Serial . print ( '\n', BYTE);
123
124     delay(2000);
125
126     return;
127 }
```

## 4 Durchführung Mittels PSoC

Die Realisierung des Projektes mittels des PSoC Microcontrollers wurde vom Lehrer vorgegeben. Dafür wurden zuerst Einzelprogramme erstellt, um die gewünschten Funktionalitäten einzeln zu testen. Die Einzelprogramme sollten dann zu einem Gesamtprogramm zusammengeführt werden, welches volle Funktionalität bietet.

Der verwendete Mikrocontroller ist der PSoC 5LP Bezeichner CY8C5888LTI-LP097.

### 4.1 Timer-Interrupt

Zum testen des Timer interrupts wird innerhalb der ISR das pin\_val flag gesetzt, welches innerhalb der main abgefragt und rückgesetzt wird. Die ISR muss so kurz wie möglich gehalten werden, um die Überlagerung mehrerer Interrupts zu verhindern.

#### 4.1.1 C-Code

Listing 3: Timer-Interrupt Haupt-Programm

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <stdbool.h>
4
5  #include "project.h"
6  #include "isrs.h"
7
8  int main(void){
9
10     isr_timer();
11     Timer_1_Start();
12     ISR_1m_Start();
13
14     CyGlobalIntEnable;
15
16     for (;;) {
17
18         Timer_1_ReadStatusRegister();
19         /* Check if flag is set, change leds accordingly */
20         if (pin_val){
21             LED_OUT_Write(1);
22             CyDelay(1000);
23
24             LED_OUT_Write(0);
25             pin_val = false;
26         }
27     }
28
29 }
```

Listing 4: Timer-Interrupt ISR

```

1  #include "isrs.h"
2
3  /* Initial variable values */
4  bool pin_val = false;
5
6  /* Function called by the isr. The inline keyword is used to avoid a call
7   * instruction
8   */
9  inline void isr_timer(){
10     pin_val = !pin_val;
11 }

```

#### 4.1.2 Top-Sheet

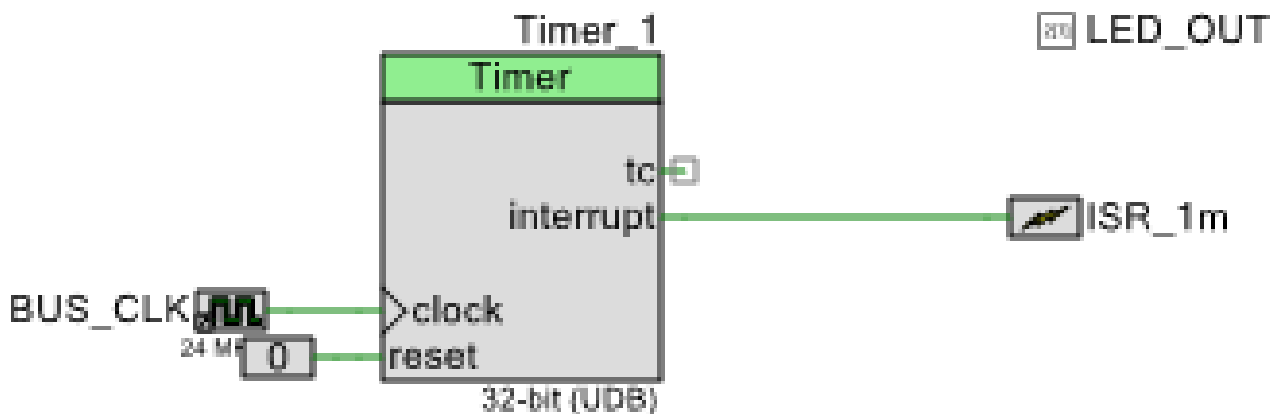


Abbildung 2: Top-Sheet des Timer-Interrupts

## 4.2 Spannungsversorgung

## 4.3 Bauteilerklärung

Es wird der IC MAX639 verwendet um die 5V Versorgungsspannung für den PSoC zu erzeugen. Der MAX639 ist dabei ein sogenannter Back- oder Step-Down-Converter welcher effizient arbeiten kann, jedoch einen Rippeln in der Spannung erzeugt. Zur Verminderung selbigen wird ein Ausgangsseitiger Kondensator verwendet.

Die maximale Eingangsseitige Spannung liegt dabei bei 11,5V, was unter der Batteriespannung ist. Um die Spannung zu reduzieren wurden daher 2 Dioden vorgeschaltet.

$R_1$  und  $R_2$  bilden einen Laut Datenblatt vorgeschriebenen Spannungsteiler welcher benötigt wird um den LOW-Battery Detector Input zu Schalten falls die Spannung 1.28V unterschreitet.

#### 4.3.1 C-Code

Listing 5: Batteriestandmessung

```

1  #include "project.h"
2

```

```
3  int main(void){
4
5      /* Hardware initialisation routines */
6      UART_Start();
7      CyGlobalIntEnable;
8
9      for (;;) {
10
11          CyDelay(1000);
12
13          /* If the digital input is beneath the threshold send out an
14             * alarm, if not, notify otherwise */
15          if (BatteryAlarm_Read() == 0){
16
17              //Does what needs to be done
18              UART_PutString("Reactor 4 is reachig critical "
19                             "temperature, cyka!\r\n");
20
21          } else if (BatteryAlarm_Read() == 1){
22              UART_PutString("Reactor 4 up and running, blyat!\r\n");
23          }
24      }
25 }
```

### 4.3.2 Top-Sheet

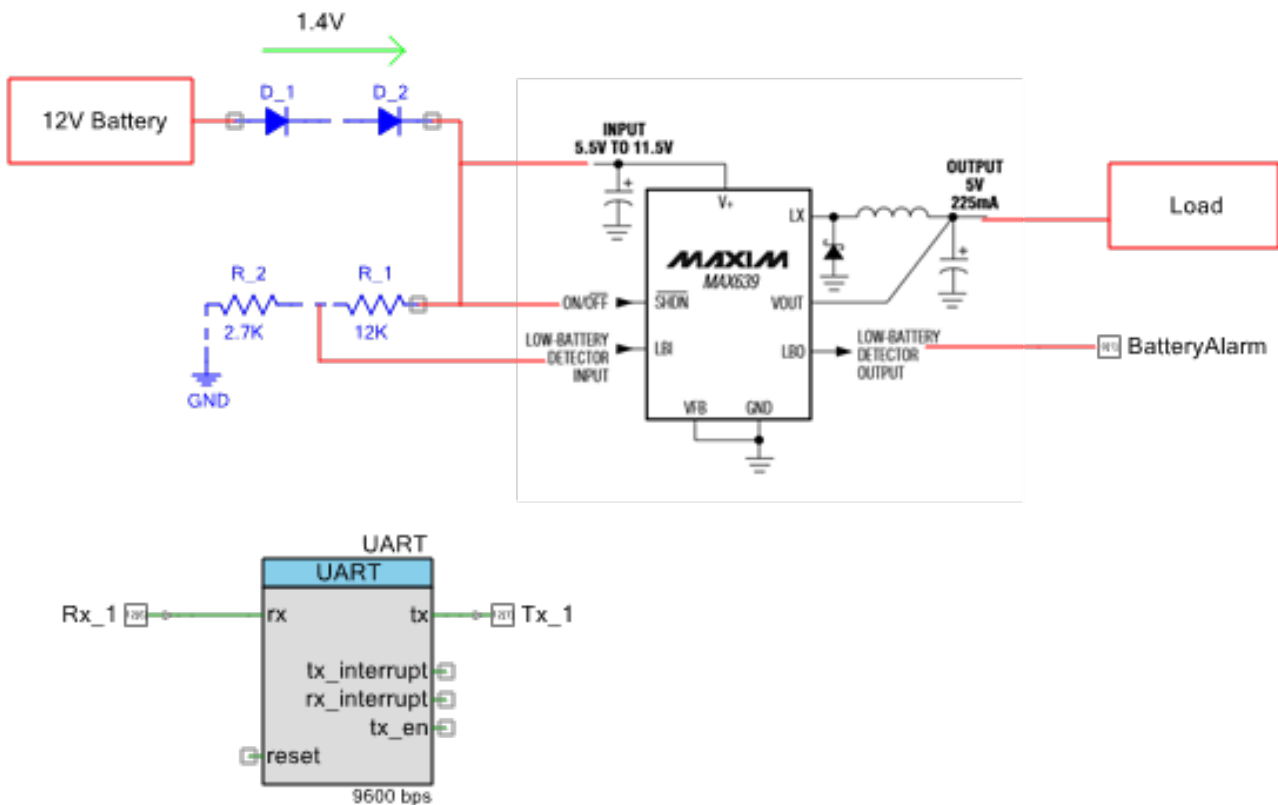


Abbildung 3: Top-Sheet des Batteriestandsmelder

## 4.4 HC-SR04

### 4.4.1 Bauteilerklärung

Der HC-SR04 ist ein IC-Baustein, der Entfernung mittels Ultraschall misst. Dafür wird nach anlegen eines min. 10µs Pulses <sup>1</sup> am TRIG Pin ein Ultraschallimpuls losgeschickt. Der Baustein rechnet sich dann aus der Laufzeit des Impuls bis zu dessen Rückkehr die Entfernung aus und gibt diese dann als PWM Signal am echo pin aus. Manche Module scheinen dabei einen Defekt zu haben oder schlichtweg billiger gebaut zu sein, da man bei diesen klar eine Kondensatorladekurve erkennen kann. Die Messergebnisse stimmen jedoch mit denen eines Funktionstüchtigen HC-SR04 überein, wenn man von CMOS Logikpegeln ausgeht, also 3.5V und höher <sup>2</sup> als WAHR annimmt. Das HC-SR04 Datenblatt behauptet hierbei jedoch, dass die Logikpegel TTL-Pegel seien, was einen kleinen Offset in diesem Falle zur Folge hätte. Dieser konnte jedoch bei uns nicht gemessen werden.

<sup>1</sup>Seite 2 HC-SR04 Datenblatt

<sup>2</sup>Halbleiter-Schaltungstechnik Seite 639

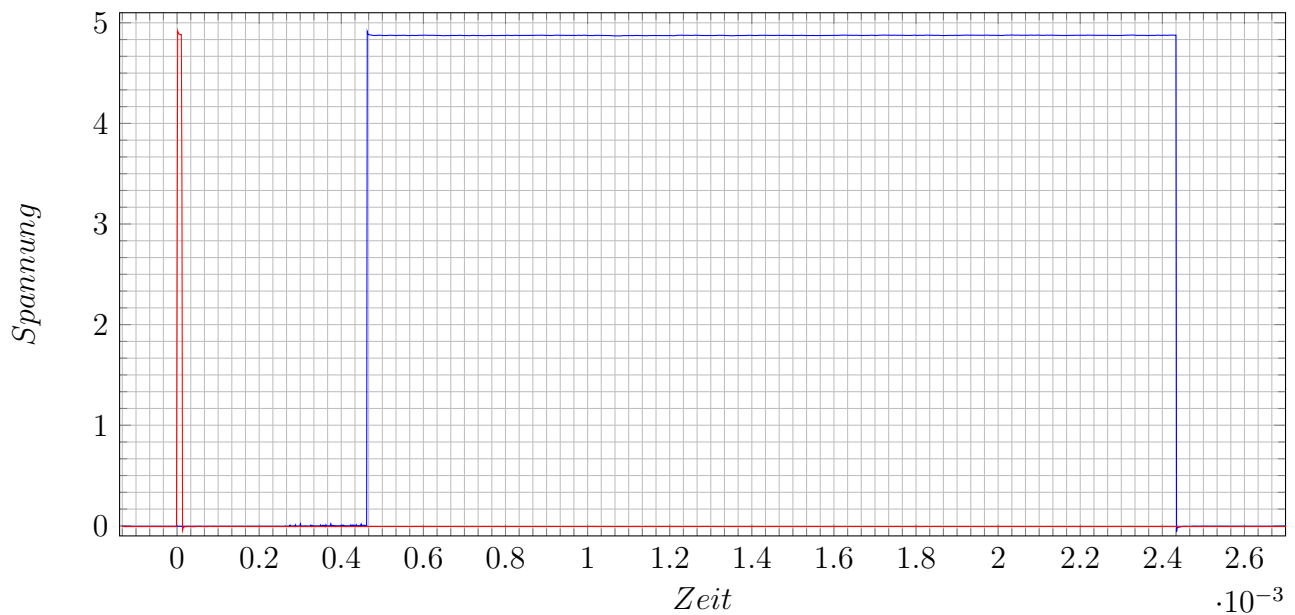


Abbildung 4: Messung mit einem funktionstüchtigen HC-SR04

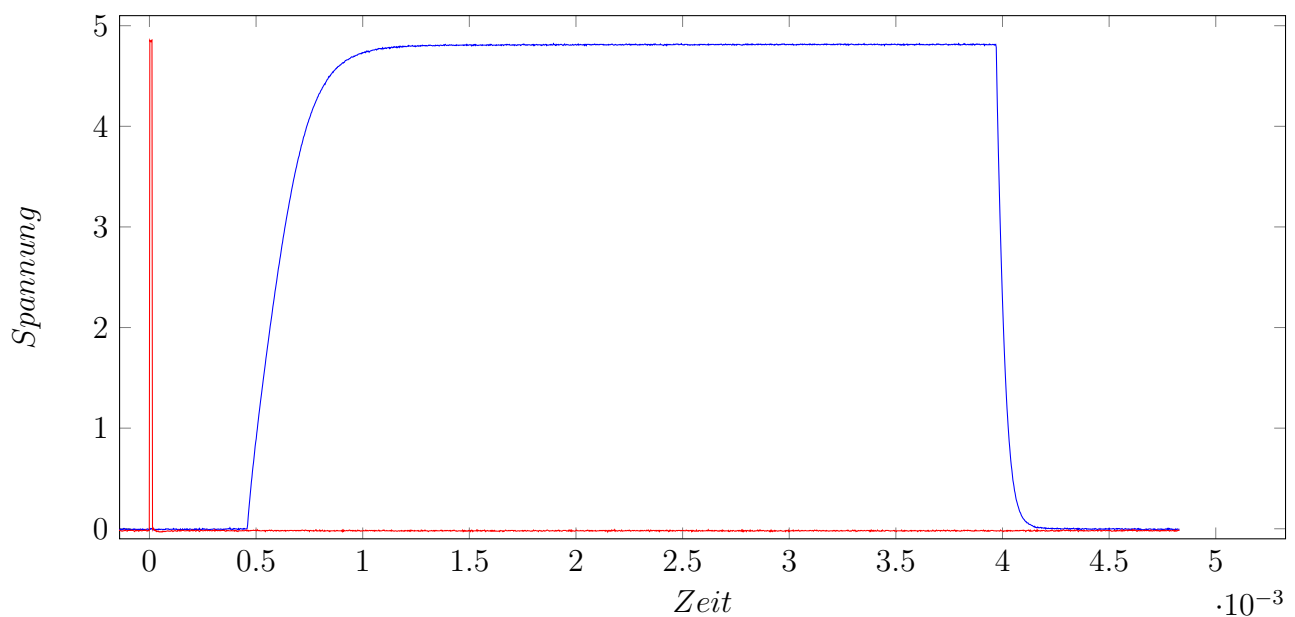


Abbildung 5: Messung mit einem sich nicht normal verhaltenden HC-SR04

Die gemessene Pulslänge in ms soll durch 57 dividiert werden um den Abstand in cm zu erhalten.

#### 4.4.2 C-Code

Programm für den Ultraschallsensor zur Messung des Wasserstands im Hochbehälter

Listing 6: PSoC funktionen für HC-SR04

```
1 /* Busy wait implementation of an HC-SR04
```

```

2  *
3  */
4  #include "hcsr04.h"
5  #include "project.h"
6
7  uint32_t get_hc_sr04(){
8
9
10     /* We assumed that a linear function correcting the
11        * measurements would be enough. We took measurements at 1m and 2m
12        * and
13        * got as results the values .36m and .72m respectively. We then
14        * proceeded to put a linear function through the points (.36, 100) and
15        * (.72, 100) which resulted in a slope of 2.77777 or 25/9 and an offset
16        * of 0. Therefore we assumed an only linear distortion. We took further
17        * measurements and got errors in the range of ~1% of the real
18        * distances, which was deemed enough.
19        */
20     #define CORRECTION_FACTOR ((double)25.0/9.0)
21
22     /* If the timeout is reached, the sensor is
23        * assumed to have been disconnected
24        */
25     #define TIMEOUT 50000
26
27     /* Avoid further distortion through triggered interrupts . */
28     CyGlobalIntDisable;
29
30     /* Place your application code here. */
31     Trigger_Impuls_Write(1);
32     CyDelayUs(10);
33     Trigger_Impuls_Write(0);
34
35
36     for(int i = 0; i < TIMEOUT && !Echo_Input_Read(); i++){
37         CyDelayUs(1);
38     }
39
40     uint32_t counter;
41
42     for(counter = 0; Echo_Input_Read() == 1 && counter <= TIMEOUT; counter
43         ++){
44         CyDelayUs(1);
45     }
46
47     CyGlobalIntEnable;
48
49     /* This statement performs an integer to float conversion and an
50        * float to integer conversion. The calculation itself happens as

```



```

50      * (double precision) float .
51      */
52      return (counter * CORRECTION_FACTOR);
53  }
54
55  uint32_t timer_us_hcsr04 = 0;
56
57  int init_hcsr04(){
58
59      return 0;
60  }

```

#### 4.4.3 Top-Sheet

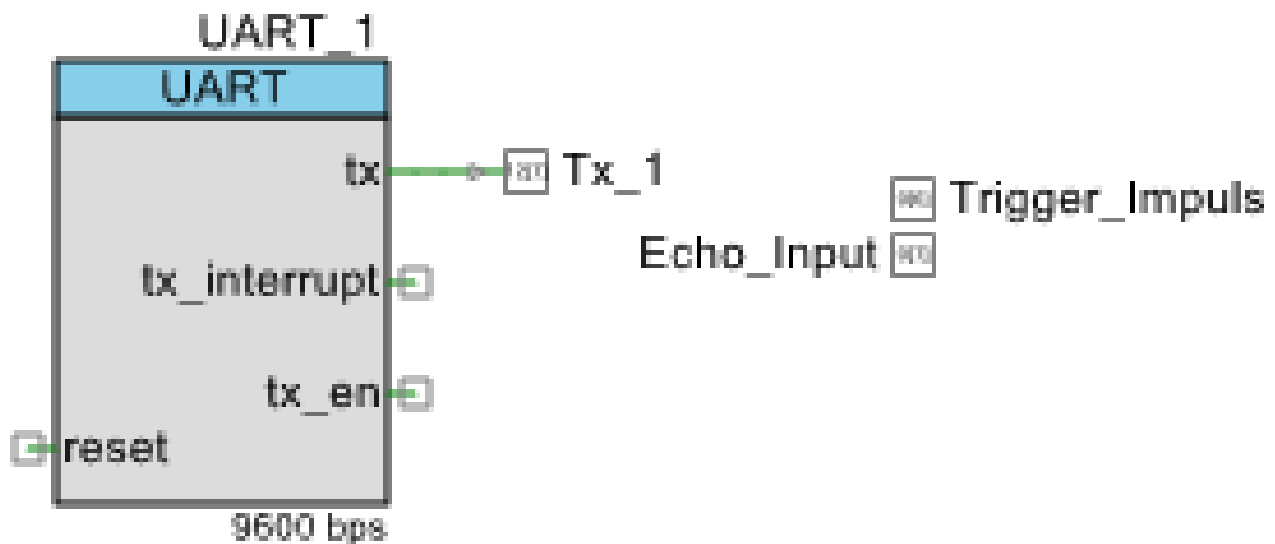


Abbildung 6: Top-Sheet der Wasserstandsmessung

## 4.5 Durchflussmessung

### 4.5.1 Bauteilerklärung

Die Messung des Wasserdurchflusses wird mittels eines Standard Wasserzählers durchgeführt. In diesem befindet sich ein Reed-Schalter welcher bei jeder vollen Umdrehung des sich im Zähler befindenden Schaufelrades einman betätigt wird. Dies entspricht einem dem Datenblatt entnehmbaren Volumen an Wasser. In unserem Falle konnte jedoch kein Datenblatt aufgefunden werden, anstelle dessen, war jedoch die Menge an Wasser pro Impuls am Gerät selbst angeschrieben.



Abbildung 7: Der Wasserzähler

#### 4.5.2 Prellen des REED-Kontaktes

Da der Reed-Schalter ein physischer Schalter ist, welcher ein zwei Metallische Oberflächen aufeinander aufschlagen lässt entsteht ein Schalter-Prellen. Mithilfe des AnalogDiscoverys wurde versucht das Schalterprellen zu ermitteln, wie man in der Grafik 4.5.2 erkennen kann prellt der Schalter nicht merklich, weshalb auf ein Entprellen mittels Kondensator, oder in Software verzichtet wurde.

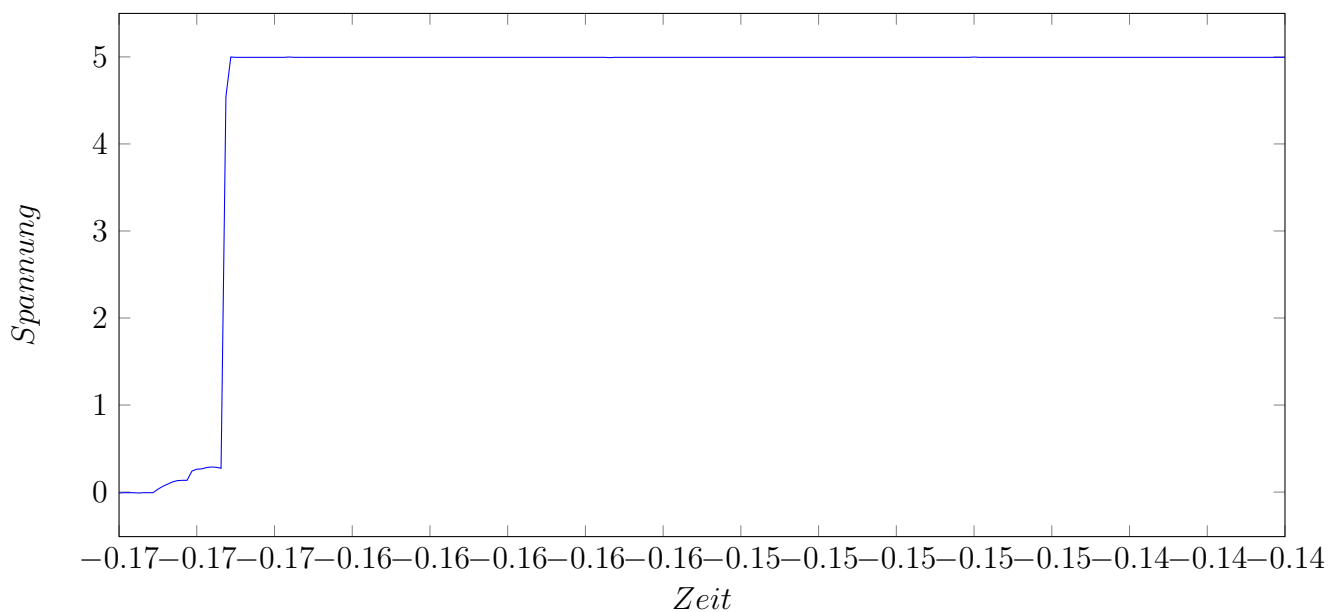


Abbildung 8: Messung des Schalterprellens mithilfe des Analog-Discovery

### 4.5.3 C-Code

Listing 7: Durchflussmessung

```

1  #include "project.h"
2  #include <string.h>
3  #include <stdio.h>
4
5  int main(void){
6
7      /* Initialize Hardware */
8      water_counter_Start();
9      UART_Start();
10     UART_PutString("INIT\r\n");
11
12     /* After initialisation enable global interrupts */
13     CyGlobalIntEnable;
14
15     for (;;) {
16         /* Delay to empty buffer */
17         CyDelay(1);
18         char buffer[30];
19         memset(buffer, 0, sizeof(buffer));
20
21         /* Print out current counter states to the user */
22         snprintf (buffer, sizeof(buffer), "Stand: %lu0L\r\n",
23                 water_counter_ReadCounter());
24         UART_PutString(buffer);
25     }

```

```

26      /* Unreachable */
27  }

```

#### 4.5.4 Top-Sheet

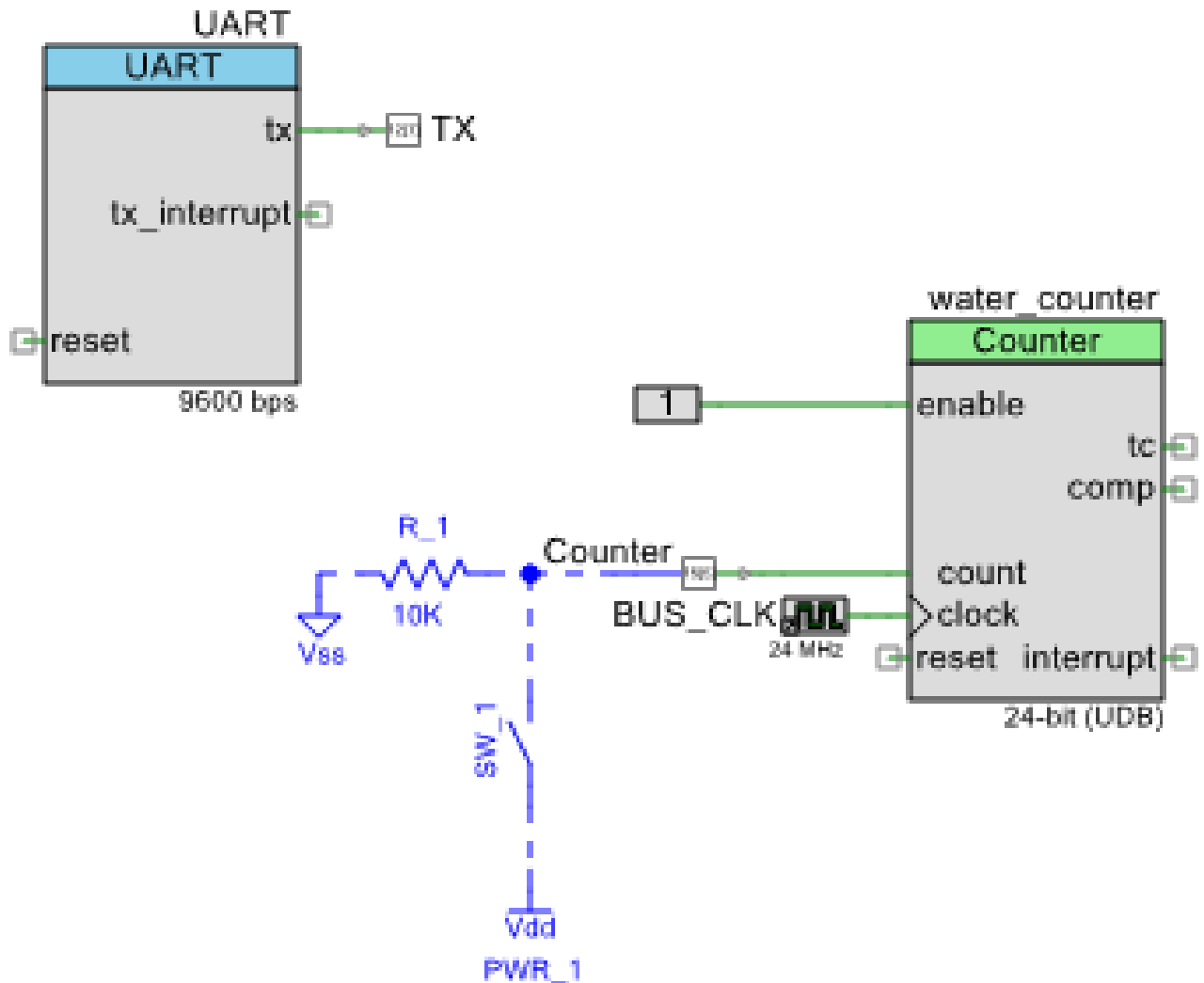


Abbildung 9: Top-Sheet de Wasserdurchflussmessung

## 4.6 DS1722

### 4.6.1 Bauteilerklärung

Der DS1722 ist ein digitaler Temperatur-Mess-IC von MAXIM Integrated. Er hat einen Messbereich von  $-60^{\circ}\text{C}$  bis  $+120^{\circ}\text{C}$  welchen er wahlweise mit 8, 9, 10, 11 oder 12 bit auflösen kann. Als Schnittstellen hat er wahlweise 3-Wire oder SPI.<sup>3</sup> Welches Protokoll verwendet wird ist abhängig vom Spannungspegel am SERMODE Pin. Wenn am SERMODE Pin 5V anliegen, wird SPI verwendet, wenn 0V anliegen wird 3-Wire verwendet, den Pin nicht zu verbinden führt zu einem undefiniertem Verhalten.

Im SPI-Modus werden die Pins SDI (Serial Data In) zu MOSI (Master Output Slave Input) und SDO (Serial Data Out) zu MISO (Master Input Slave output). Der Chip-Enable Eingang<sup>4</sup> ist HIGH-Aktiv, was im PSoC ein vorschalten eines Inverters vor den CE-Pin erforderlich macht.

Aufgrund anfänglicher Probleme mit dem PSoC wurde die Machbarkeit mittels ATMega nochmals überprüft. Dafür wurde eine ATMega2560 als SPI Master verwendet, auf welchem die SPI-Pins in der GPIO-Bank B sind. Abbildung 4.6.1 zeigt einen mittels ATMega 2560 erzeugten Datenverkehr mit dem DS1722.

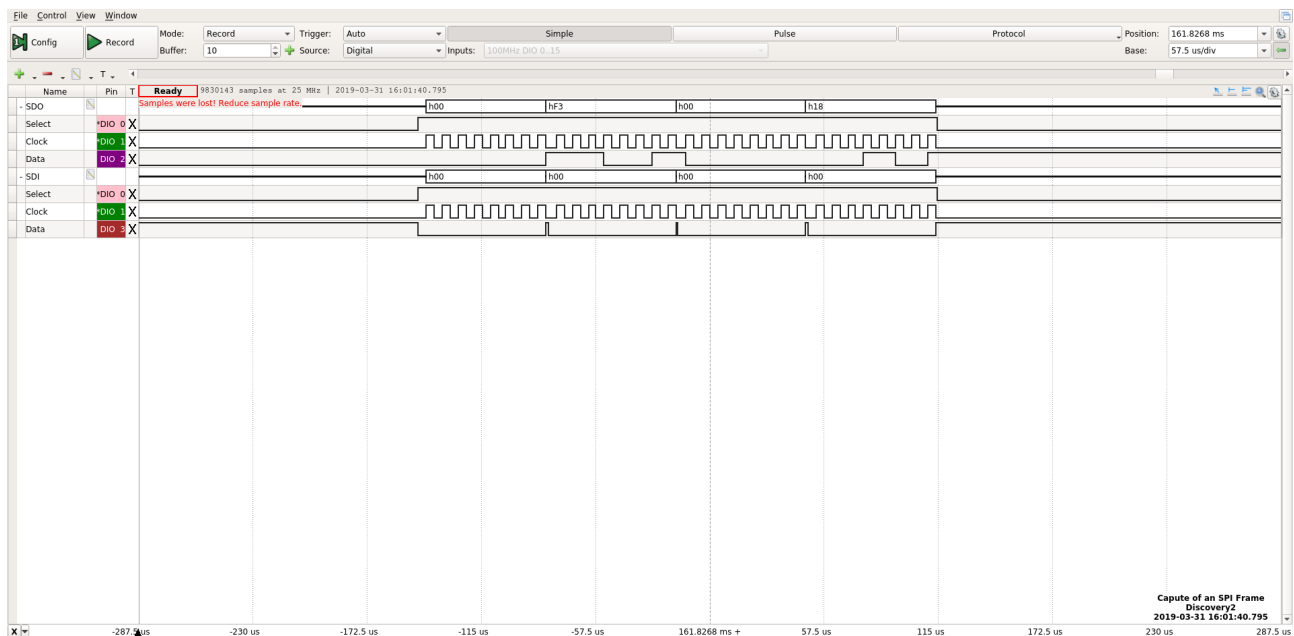


Abbildung 10: Erzieltes Ergebnis mit ATMega 2560

Die oben dargestellte Datensequenz entspricht erwarteten Werten. Die funktionstüchtigkeit des Sensors wurde dadurch bestätigt, und die Umsetzung mittels PSoC wurde erneut in Angriff genommen. Die Temperatur-Messung mittels des Codes in Listing 8 ist der aktuelle Stand.

<sup>3</sup>Serial Peripheral Interface, Standard von Motorola

<sup>4</sup>Der CE-Pin wird manchmal auch SS (Slave Select) bezeichnet

### 4.6.2 C-Code für DS1722

Listing 8: Fehlerhaftes Temperaturmessungsprogramm für den DS1722

```
1
2 #include "project.h"
3
4 #include <stdint.h>
5 #include <stdio.h>
6
7 int main(){
8
9     SPI_Start();
10    UART_Start();
11    CyGlobalIntEnable;
12
13    /* Write initial configuration into specified registers */
14    SPI_WriteTxData(0x80);
15    SPI_WriteTxData(0xE9);
16    CyDelay(10);
17
18    /* Empty the RX buffer */
19    while(SPI_GetRxBufferSize() == 0){}
20    SPI_ReadRxData();
21    while(SPI_GetRxBufferSize() == 0){}
22    SPI_ReadRxData();
23
24    UART_PutString("INIT\r\n");
25
26    for (;;) {
27        /* Fill the SPI TX buffer in order to read the first 3 bytes from the
28        * first 3 registers
29        */
30        SPI_WriteTxData(00);
31        SPI_WriteTxData(00);
32        SPI_WriteTxData(00);
33        SPI_WriteTxData(00);
34        while(SPI_GetRxBufferSize() == 0){/* Wait for rx buffer to be ready */}
35        uint8_t recv1 = SPI_ReadRxData();
36
37        while(SPI_GetRxBufferSize() == 0){ /* Wait for rx buffer to be ready */}
38        uint8_t recv2 = SPI_ReadRxData();
39
40        /* Print out received data to user */
41        char buf[100];
42        memset(buf, 0, sizeof(buf));
43        snprintf (buf, sizeof(buf), "Recv1: %X\r\n", recv1);
44        UART_PutString(buf);
45        memset(buf, 0, sizeof(buf));
46        snprintf (buf, sizeof(buf), "Recv2: %X\r\n", recv2);
```

```

47     UART_PutString(buf);
48
49 }
50
51 }

```

### 4.6.3 Top-Sheet

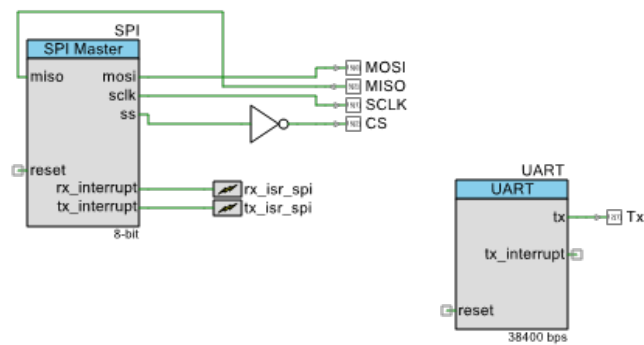


Abbildung 11: Top-Sheet der Aktuellen DS1722 Schaltung

## 4.7 BIM2

### 4.7.1 Bauteilerklärung

Das BIM2 Modul ist ein FM-Transceiver welcher auf 433MHz sendet und empfängt. Es kann eine Maximale Übertragungsgeschwindigkeit von 160Kb/s und hat dabei eine Reichweite von 200m. Mithilfe einer Empfangsanlage mit Richtwirkung kann diese Reichweite jedoch nochmals erhöht werden. Das BIM-Modul ermöglicht eine Halb-Duplex Übertragung, was bedeutet, dass nur 1 Teilnehmer reden darf.

### 4.7.2 Anschluss

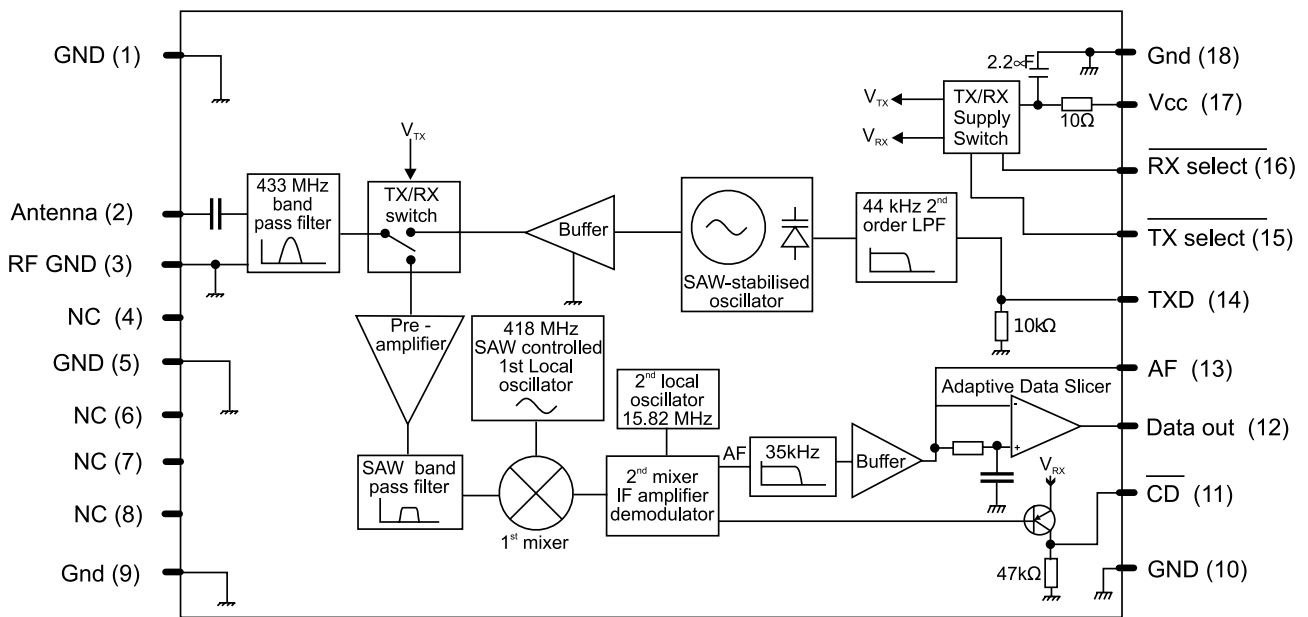


Abbildung 12: Ersatzschaltbild des BIM2 laut Datenblatt

Jedes BIM2 Modul kann sowohl als Receiving-End als auch als Transmitting-End dienen. Zum auswählen muss entweder RX-Select oder TX-Select auf GND gezogen werden, und die nicht benötigte Leitung auf  $V_{CC}$ . Dann kann entweder, je nach Auswahl, an TXD Daten versendet, oder an Data out Daten empfangen werden. Die folgenden Beispielübertragungen wurden mittels PSoC durchgeführt.

### 4.7.3 C-Code für BIM2

Listing 9: Übertragungstestdes BIM2

```

1  /* BIM-2 Testprogramm */
2
3  #include "project.h"
4
5  int main(){
6
7      /* Hardware initialisation */
8      UART_Start();
9
10     /* Set the ARM's global interupt enable flag */
11     CyIntEnable;
12
13     UART_PutString("INIT\r\n");
14
15     for (;;) {
16         /* Print output to the bim module */
17         UART_PutString("UU");
18         CyDelay(100);
19     }

```



20  
21  
22 }

#### 4.7.4 Übertragungsbeispiele

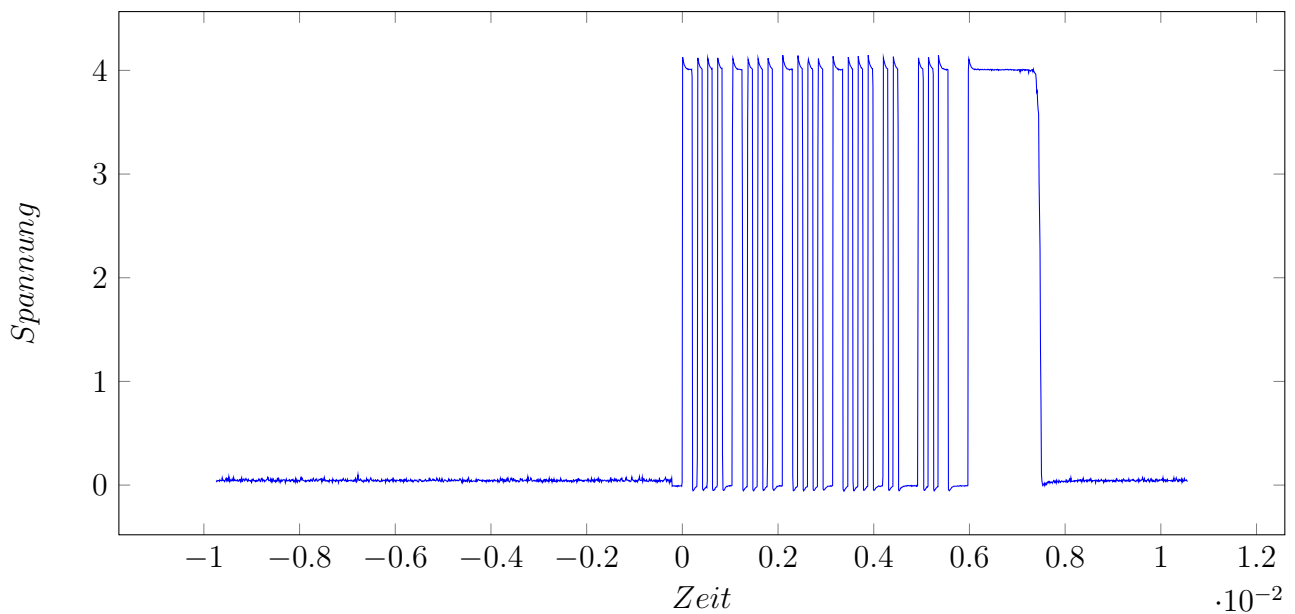


Abbildung 13: Messung eines UART Frames über eine Distanz von 10m

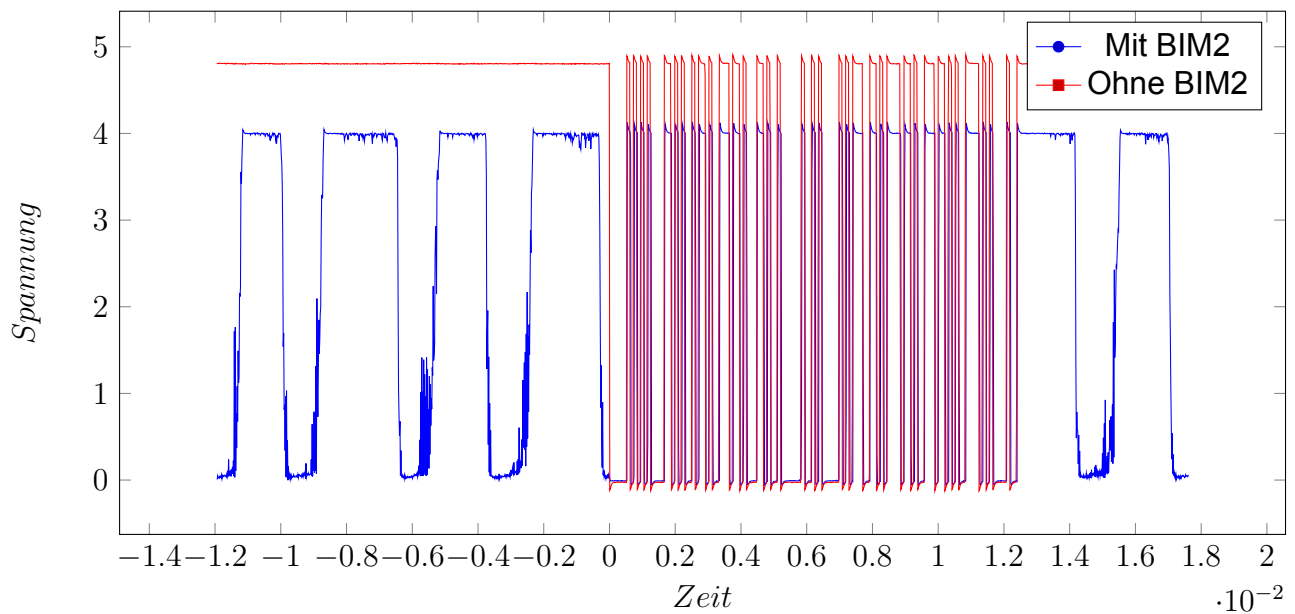


Abbildung 14: Messung des Rauschens beim Übertragen über das BIM2

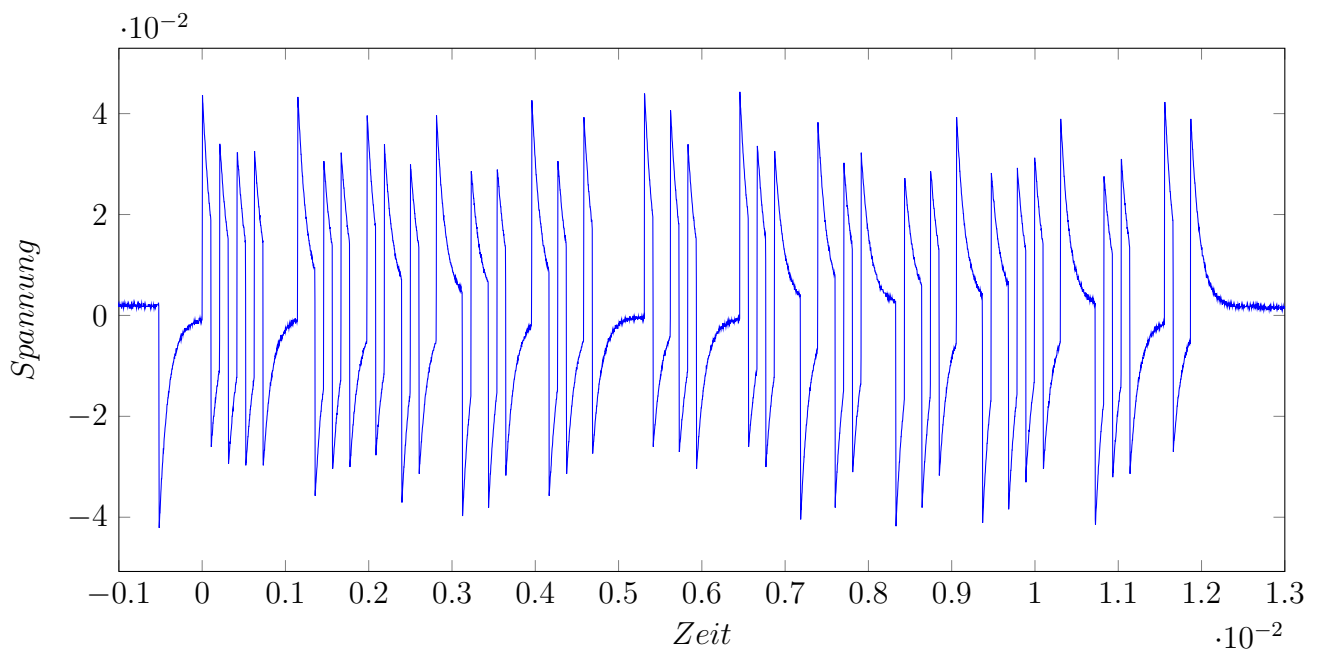


Abbildung 15: Messung des Antennenausganges des BIM2

**Conclusio** Es muss eine Art und Weise implementiert werden um die übertragenen Daten vom Umgebungsrauschen unterscheiden zu können. Dies wird durch ein Preamble realisiert welches eine Start Sequenz beinhaltet. Diese soll möglichst leicht vom Rauschen zu unterscheiden sein. Dafür wurde 3 mal 0x55 verwendet, da dies Binär 01010101 darstellt. Regelmäßige und schnelle Bitfolgen dieser Art sind in einer normalen Umgebung im Rauschen unwahrscheinlich.

## 5 Anhang

### 5.1 Literaturverzeichnis

- **HC-SR04 Datenblatt**  
Erhalten von mouser.com am 2019-03-10.
- **DS1722 Datenblatt**  
Erhalten von maximintegrated.com am 2019-03-10.
- **MAX640 Datenblatt**  
Erhalten von maximintegrated.com am 2019-03-10.
- **CY8C5888LTI-LP097 Datenblatt**  
Erhalten von cypress.com am 2019-03-10.
- **Ulrich Tietze Christoph Schenk Halbleiter-Schaltungstechnik**  
Vorliegend in 12. Auflage Springer-Verlag Berlin Heidelberg 2002
- **BIM-2 Datenblatt**  
Erhalten von radiometrix.com 2019-05-06.