Bastelhilfe: Regensensor und LoRa auf dem Arduino

Dieses Handout ist gedacht als eine kleine Hilfe, solltet ihr selbst ein Projekt basteln wollen, welches LoRa oder einen Regensensor einsetzt.

1 EINEN REGENSENSOR SELBST BAUEN

Der Regensensor kann entweder mit einem Operationsverstärker Chip oder mit diskreten Transistoren gebaut werden. Die OPV-Variante bietet ein saubereres Ausgangssignal, ist aber etwas schwieriger aufzubauen. Dazu wird folgendes benötigt:

OPV-Variante	Transistor-Variante
Operationsverstärker mit gleicher Pinbelegung wie ein LM358	2x beliebigen NPN-Transistor alternativ: 1x Darlington-NPN-Transistor
3x 10kΩ Widerstand	1x 1kΩ Widerstand
1x 100kΩ Widerstand	1x 10kΩ Widerstand
1x 1MΩ Widerstand	
Multi-Turn Potentiometer 10-100kΩ	Multi-Turn Potentiometer ca. 470kΩ
Pin-Header mit mindestens 5 Pins	
Lochraster Platine	
Stäbe aus Messing oder anderem Material welches nicht korrodiert	
Ein Behälter für die Elektroden	

Wenn ihr die OPV-Variante bauen wollt, könnt ihr einfach das Layout aus Abbildung 1 verwenden. Sollte euch die Hysterese nicht passen, spielt etwas mit den Widerständen des Schmitt-Triggers herum. Die Formeln rechts berechnen die Hysterese des Wasser-Widerstands in Abhängigkeit von R_1 (im Schaltplan 10kΩ) und R_2 (im Schaltplan 100kΩ) des Schmitt-Triggers. Der zweite OPV kann weggelassen und der Ausgang direkt an den Arduino angeschlossen werden, jedoch enthalten die meisten OPV-ICs sowieso zwei OPVs, weshalb er als Puffer verwendet wird.

$$U_{low} = \left(\frac{R_2}{R_1 + R_2} * U_{pot}\right)$$

$$U_{high} = \left(\frac{R_2}{R_1 + R_2} * U_{pot}\right) + \left(\frac{R_1}{R_1 + R_2} * 5V\right)$$

$$R_{low} = \frac{(5V - U_{low}) * 1M\Omega}{U_{low}}$$

$$R_{high} = \frac{(5V - U_{high}) * 1M\Omega}{U_{high}}$$

$$\Delta R_{hyst} = |R_{high} - R_{low}|$$

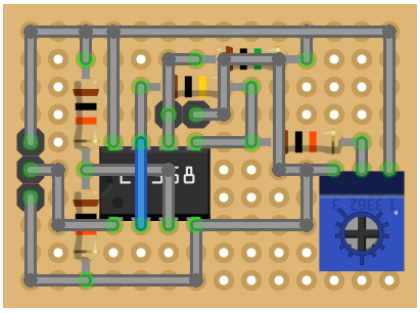


Abbildung 1

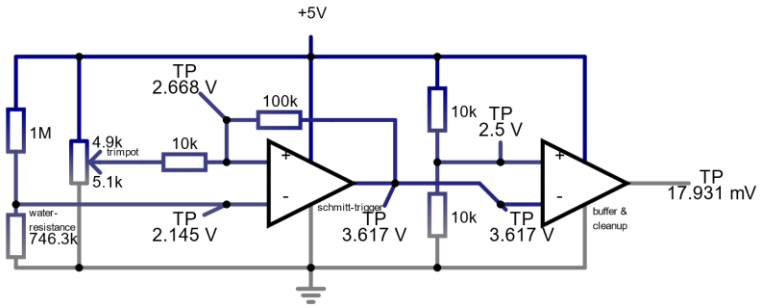


Abbildung 2

Für die Transistor-Variante habe ich kein Layout erstellt, nur einen Schaltplan (Abbildung 3). Die Bauteilanzahl ist so gering, dass man die Schaltung sogar ohne Platine – die Bauteile einfach direkt an den Beinchen verbunden – aufbauen könnte.
Der Regensammler/-detektor kann aus einem beliebigen wetterfesten Behälter und Elektroden gebastelt werden.

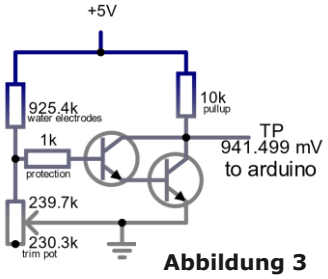


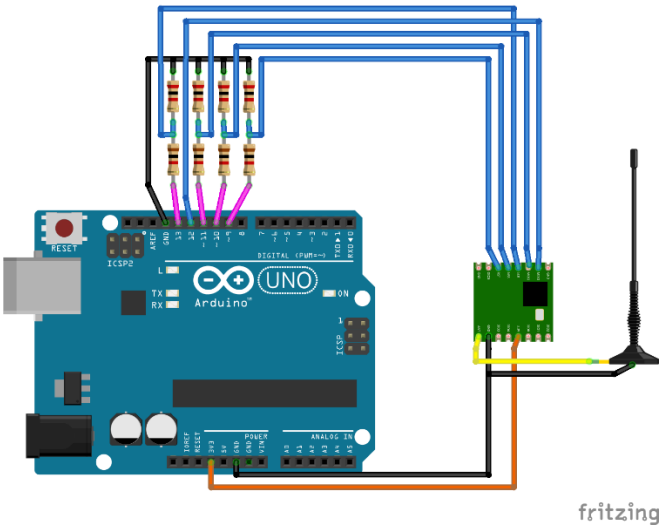
Abbildung 3

2 LORA GRUNDLAGEN IN ARDUINO

Eine kurze Einführung in LoRa auf dem Arduino, falls ihr selbst LoRa in einem Arduino Projekt verwenden möchtet, um zwischen zwei oder mehr Geräten direkt (Peer-to-Peer) zu kommunizieren. Grundlegende Arduino-Kenntnisse sind vorausgesetzt.

2.1 HARDWARE

2.1.1 ARDUINO



Der Kern des LoRa-Moduls ist ein SX1276 Chip welcher nur 3,3V Pegel verträgt, jedoch benutzt der Arduino 5V Pegel. Da SPI – das Protokoll zwischen LoRa Chip und Arduino – keine bi-direktionalen Datenleitungen (also Leitungen, bei denen beide Seiten senden würden) hat, ist die Umwandlung der vom aus Arduino sendenden Leitungen aber leicht durch Spannungsteiler möglich, während die empfangende Leitung MISO (**M**aster **I**n, **S**lave **O**ut) direkt verbunden wird. Der Spannungsteiler-Widerstand Richtung GND sollte ungefähr den doppelten Wert des Widerstands in Richtung Arduino haben, um die Spannung von 5V auf 3,3V zu wandeln. Dies wird durch die Spannungsteiler-Formel bewiesen.

$$U_{out} = \frac{R_1}{R_1 + R_2} * U_{in} = \frac{20k\Omega}{20k\Omega + 10k\Omega} * 5V = 3.\overline{3}V$$

Bei mit 3,3V betriebenen Mikrocontrollern wie dem ESP8266 oder ESP32 sind die Spannungsteiler natürlich nicht nötig. Möchte man die asynchronen Funktionen der Bibliothek nutzen, muss man zusätzlich DIO0 des Modules an einen Interrupt-fähigen Pin (2 oder 3 auf dem Arduino UNO) anschließen.

2.1.2 ANTENNEN

Antennen sind eine komplizierte Angelegenheit. Es gibt zahlreiche unterschiedliche Typen, die sich je nach Aufgabe besser oder schlechter eignen. Sie müssen aber alle auf die Frequenz und den Sender abgestimmt sein. Funk- und Antennentheorie sind ein sehr kompliziertes Gebiet, weshalb ich hier nur kurz auf das Thema eingehe.

Ein guter und leicht zu bauender Typ von Antenne ist eine ¼-Lambda-Ground-Plane. Sie ist nicht ganz rundstrahlend, sendet somit keine Energie in den Boden oder das Weltall, besteht nur aus 5 Stücken Draht von richtiger Länge und lässt sich mit diesem Online-Rechner berechnen: https://m0ukd.com/calculators/quarter-wave-ground-plane-antenna-calculator/

Meistens werden mit den LoRa-Modulen auch schon Antennen mitgeliefert. Diese sind passabel für kürzere Strecken.

2.2 SOFTWARE

Man kann und muss nicht immer alles komplett selbst machen. Es benötigt viel Zeit, ist kompliziert, aber nicht immer nötig, da viele komplexe Dinge wie z.B. die Ansteuerung eines LoRa Moduls auch schon einmal von jemand anderem realisiert worden sind. Veröffentlicht dieser Jemand seine getane Arbeit als eine Code-Bibliothek für andere zum Nutzen, muss man nicht unnötigerweise die gleiche Arbeit noch einmal machen. Für Kleinigkeiten ergibt das wenig Sinn – oft ist es schneller und auch lehrreicher, wenn man es selbst erledigt – doch für Größeres kann man sich auch auf jemand anderen verlassen. Natürlich übernimmt man so auch alle etwaigen Fehler des anderen. Ein solides Projekt auf einem wackeligen Fundament wird früher oder später zusammenbrechen und mit jeder Bibliothek steigt die Chance ein marodes Teil einzubauen, wenn man nicht aufpasst.

Für LoRa Module gibt es einige Bibliotheken. Ich habe die *„arduino-LoRa"* Bibliothek (https://github.com/sandeepmistry/arduino-LoRa/) benutzt, da diese macht, was ich brauche und nicht mehr. Die Benutzung ist leicht zu verstehen und es wird nicht Prorammspeicher auf dem Arduino verbraucht.

Eingebunden wird die Library mit `#include <LoRa.h>` am Anfang eines Sketches.

2.2.1 SETUP

Bevor man loslegen kann, muss man die Pins, an denen das Modul angeschlossen ist, festlegen, es initialisieren, und in den Bereitschaftszustand setzen.

```
LoRa.setPins(ss, reset, dio0);
LoRa.setSPIFrequency(1E6);
LoRa.begin(frequency);
LoRa.idle();
```

Der Reset-Pin kann auf einen beliebigen Pin – standardmäßig Pin 9 – gelegt werden. Falls die Pins am Arduino knapp werden, kann der RESET Pin vom Modul auch an das RESET Signal das Arduinos angeschlossen werden. In dem Fall setze `reset` auf -1. Der `dio0` Pin wird auf -1 gesetzt, da dieser nur im asynchronen Modus der Library, welcher hier nicht beschrieben wird, als eingehender Interrupt genutzt wird. Der `ss` Pin kann auf irgendeinen Pin gelegt werden – standardmäßig Pin 10.

`frequency` wird auf die gewünschte Frequenz in Hertz gesetzt, z.B. `4339E5` für 433,9MHz. Module sind für eine ungefähre Frequenz ausgelegt, von dieser man nicht zu weit abweichen sollte, um die beste Leistung zu erreichen. Auch muss beachtet werden auf welchen Bändern man überhaupt senden darf. Mitten im FM Radio Band oder auf der Frequenz einer Mobilfunkzelle hat man nichts zu suchen, wenn man keinen Ärger mit der Bundesnetzagentur will. Durch euch nutzbare Frequenzen finden sich in Kapitel 3.3.

Die Verbindung zwischen LoRa Modul und Arduino wird via SPI hergestellt. Die Code-Bibliothek setzt die Übertragungsgeschwindigkeit standardmäßig auf 10MHz. Das ist sehr schnell – oftmals zu schnell – obwohl hier nur eine Handvoll Bytes pro Sekunde übertragen wird. Um Fehler, die z.B. durch längere Leitungen entstehen können zu verhindern, ist es Vorteilhaft, diese Übertragungsgeschwindigkeit herunterzusetzen. In den Beispiel-programmen wird sie mit `LoRa.setSPIFrequency(1E6)` auf 1MHz aka 10^6 Hertz gesetzt.

2.2.2 PARAMETER

LoRa hat einige Parameter mit denen man zwischen Übertragungsgeschwindigkeit und Störungsresistenz. Verbessert man eine Eigenschaft, verschlechtert sich die andere. Bis auf die Coding Rate und CRC müssen die Parameter auf beiden Seiten gleich eingestellt sein.

```
LoRa.setTxPower(txPower);
LoRa.setSpreadingFactor(spreadingFactor);

LoRa.setSignalBandwidth(signalBandwidth);
LoRa.setCodingRate4(codingRateDenominator);
LoRa.enableCrc(); //LoRa.disableCrc();
```

2.2.2.1 BANDBREITE

`signalBandwidth` setzt die beanspruchte Bandbreite. Sie ist entweder gesetzlich oder durch die Hardware begrenzt. Die verwendeten LoRa Chips können bis zu 500kHz Bandbreite des Spektrums benutzen, müssen jedoch innerhalb des Bandes bleiben und dürfen deshalb gesetzlich nicht immer die volle Bandbreite benutzen. Man sollte die Bandbreite nur so hoch wie nötig setzen, da eine hohe Bandbreite zwar die Geschwindigkeit erhöht, aber die Empfindlichkeit des Empfängers – wie schwach ein Signal sein kann, bevor es nicht mehr dekodierbar ist – leicht herabsetzt. Man sollte sie jedoch auf den für Bastler erhältlichen Modulen nicht unter 62,5kHz setzen, da diese keinen ausreichend stabilen Oszillator haben, um zuverlässig bei kleineren Bandbreiten zu funktionieren. Da sie in bestimmten Schritten einstellbar ist, bleiben so nur 62,5kHz, 125kHz, 250kHz, und 500kHz.

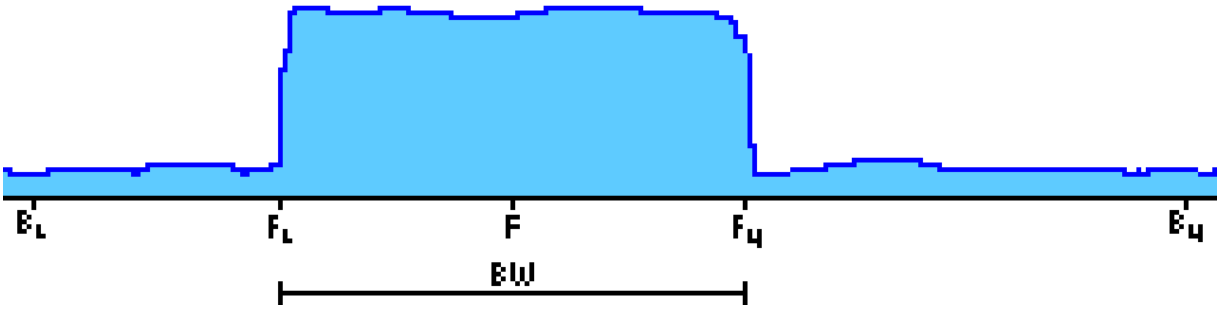


Abbildung 4

In der Mitte des oberen (F_u) und unteren (F_l) Ende der genutzten Bandbreite liegt die ausgewählte Frequenz *F*. Rechtlich darf keines der Enden über die Grenzen eines erlaubten Bandes (B_l, B_u) herausragen. Dabei muss man beachten, dass die Frequenz tatsächlich etwas abweichen kann. Dieser Frequenzfehler (FE) beträgt erfahrungsgemäß 5kHz – 10kHz. Im Beispiel wird mit einem Worst-Case Szenario von 20kHz gerechnet.

$$F_l = F - \left(\frac{BW}{2}\right) - FE$$
$$F_u = F + \left(\frac{BW}{2}\right) + FE$$
$$OK = (B_l < F_l) \wedge (F_u < B_u)$$

$$F_l = 433{,}9MHz - \left(\frac{500kHz}{2}\right) - 20kHz = 433{,}63MHz$$
$$F_u = 433{,}9MHz + \left(\frac{500kHz}{2}\right) + 20kHz = 434{,}17MHz$$
$$OK = (433{,}05MHz < F_l) \wedge (F_u < 434{,}79MHz) = wahr$$

2.2.2.2 SENDELEISTUNG

`txPower` stellt die Sendeleistung in dBm ein. Die Sendeleistung ist jene Leistung, die vom Chip aus in die Antenne ausgegeben wird. Dies sollte man nicht mit der Effektiven Strahlungsleistung (ERP) verwechseln, welche Verluste in Komponenten und den Effekt der Antenne, die Sendeleistung auf einen kleineren Bereich zu fokussieren, mit einberechnet. In Abbildung 5 ist ein Sender mit der gleichen Sendeleistung, aber zwei unterschiedlichen Antennen abgebildet. Die eine sendet in alle Richtungen, die andere sendet nur in einem Viertel des Bereiches und konzentriert somit Leistung. Je nach Antennentyp ist diese Konzentrierung unterschiedlich stark, bei den geringen Sendeleistungen wie wir senden dürfen aber recht vernachlässigbar, solange man keine Richtantennen verwendet. In Kapitel 3.3 sind die maximal erlaubten Strahlungsleistungen angegeben. Wer sicher sein möchte, zieht bei nicht ganz rundstrahlenden Antennen den maximale Gewinn („Gain") der Antenne von der Sendeleistung ab.

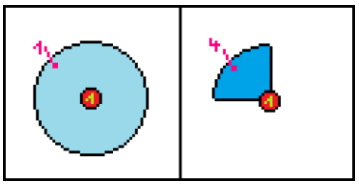


Abbildung 5

2.2.2.3 SPREADING FACTOR

LoRa verwendet steigende und fallende Töne, sogenannte Chirps, um Daten zu übertragen. Der Spreading-Factor, kurz SF, verändert auf wie viele dieser Chirps ein Bit an Daten verteilt wird. Je höher der SF, desto empfindlicher ist der Empfänger, doch die Übertragungsgeschwindigkeit nimmt ab.

2.2.2.4 CODING RATE

Die Coding Rate, kurz CR, legt den Anteil an Redundanz fest, der in das Signal eingebaut wird. Dies ist besonders in Gegenden mit viel Rauschen und Störungen auf dem Band nützlich, um zuverlässige Übertragung sicher zu stellen. Sie ist von ⅘ bis ⁴⁄₈ Redundanz-anteil einstellbar. Je mehr Redundanz man einbaut, desto weniger Zeit bleibt natürlich für die Daten. Auch hier tauscht man Störungsresistenz gegen Übertragungsgeschwindigkeit.

2.2.2.5 CRC

Die CRC ist eine Prüfsumme, die mit in die Pakete eingebaut werden kann und automatisch vom Modul geprüft wird. Wenn die Prüfsumme inkorrekt ist, wird das Paket einfach von der Code-Bibliothek ignoriert.

2.2.2.6 RESULTIERENDE DATENRATE

Um die Übertragungsgeschwindigkeit zu errechnen, gibt es eine Formel, oder für die Faulen, Rechner im Internet. Man bedenken das auch „unsichtbare" Teile eines LoRa Paketes wie der Header oder die CRC ein paar Bits benötigen und die Übertragung so etwas länger dauern kann als erwartet.

Datenrate in Bits pro Sekunde (durch 8 Teilen für Bytes pro Sekunde) (1):

$$R_{bit} = SF * \frac{\left(\frac{4}{4+CR}\right)}{\left(\frac{2^{SF}}{BW}\right)} * 1000$$

Online-Rechner: https://unsigned.io/understanding-lora-parameters/

Quellen für Kapitel „2.2.2 Parameter": (1; 2; 3)

2.2.3 SENDEN

Bevor man überhaupt etwas Senden kann, muss man sich erst einmal überlegen auf was man Wert legt und die vorherigen Parameter dementsprechend einstellen. In den Beispielprogrammen werden einfach die Standartwerte der Bibliothek verwendet.

LoRa versendet Daten in Paketen, welche maximal 256 Bytes lang sein können. Deshalb muss man, bevor etwas gesendet werden kann, zuerst überlegen *was genau wie* gesendet werden soll. Es ist ratsam das erste Byte für eine „Magische Zahl" zu nutzen, welche das Projekt von anderen auseinanderhält, die möglicherweise die gleichen LoRa Parameter und hoffentlich eine andere Zahl verwenden. Wenn man mehrere Dinge steuern möchte, ist es am einfachsten, das nächste Byte als eine Befehlsnummer zu verwenden mit der beschrieben wird, wie die nachfolgenden Bytes verarbeitet werden sollen. Um den Code besser lesbar zu halten, ist es sinnvoll mit einem `enum` Zahlen einen Namen zuzuweisen. Dies muss nicht von Hand geschehen, doch in den Bespielprogrammen sind zum besseren Verständnis alle Namen manuell zugewiesen. Würde man dies weglassen, würden die Namen genauso von 0 aufwärts nummeriert. Falls man „verschluckte" oder doppelte Pakete erkennen und/oder eine Neuübertragung derer anfragen möchte sollte man das Byte vor oder nach der Befehlsnummer als eine Paket ID nutzen, die Pakete klar auseinanderhält (dies ist der Einfachheit halber nicht in den Beispielprogrammen beschrieben). Danach folgen die Datenfelder, die man versenden möchte. Es ist sinnvoll, alle Datenfelder des Paketes auf der seriellen Konsole auszugeben, um nachvollziehen, ob das Programm arbeitet, wie es sollte und falls nicht, die Fehlersuche zu erleichtern.

Man kann jedem Byte ein Datenfeld zuordnen, doch oft ist es sparsamer ein Byte in mehrere Abschnitte zu teilen. Dies ist durch bitweise Operationen wie UND (&), nach links verschieben (<<) und nach rechts verschieben (>>) möglich. Dadurch kann man ein Byte in mehrere Datenfelder teilen oder jedes Bit einzeln nutzen, falls man mehrere Datenfelder mit nur zwei Zuständen senden möchte. Wenn es notwendig ist, mehrere Byte große Werte zu senden, kann man dazu eine `union` verwenden. Diese kann den gleichen Bereich im Speicher als verschiedene Datentypen verfügbar machen. So kann man die einzelnen Bytes eines größeren Datentyps in das LoRa Paket schreiben und daraus heraus wieder Stück für Stück zurück in den Datentyp lesen. Ein Fallstrick dabei ist, dass verschiedene Prozessorarchitekturen mehrere Byte lange Werte in anderer Reihenfolge speichern. 12345 wird auf „big endian" Architekturen als `30 39` gespeichert während „little endian" Architekturen es als `39 30` speichern. Die meisten Microcontroller sind „big endian", doch sollte man einmal das Problem haben, kann man die Bytes einfach in umgekehrter Reihenfolge in das Array der `union` schreiben.

Siehe Beispielprogramm 1 im Anhang zum Thema Daten versenden. Es wird angenommen, dass beide Systeme die gleiche „endianess" haben. Die folgenden Befehle werden in beiden Beispielprogrammen genutzt:

Name	„Magische Zahl"	Befehls-nummer	Daten Byte 1	Daten Byte 2	Daten Byte 3
LED-Helligkeit	0xAC	0x00	Helligkeit 8bit (0-255)		
Text Ausgeben	0xAC	0x01	Text von unbestimmter Länge (Jedes Byte ein Zeichen, wenn Text ASCII Kodiert)		
Ton Abspielen	0xAC	0x02	Tonhöhe in Hertz 16bit (0-65535)		Länge in ms*10 8bit (0-255)
Zwei Zahlen	0xAC	0x03	X 3bit | Y 5bit		

2.2.4 EMPFANGEN

Zum Empfangen setzt man das LoRa Modul mit `LoRa.receive()` in den Empfangsmodus und fragt regelmäßig mit `LoRa.parsePacket()`, ob es denn ein Paket empfangen hat und wenn ja, wie lang es ist. Wenn ein Paket empfangen wurde, muss man es Byte für Byte vom Modul lesen. Es macht Sinn, das Paket in einem Array gleich lang wie das Paket (oder länger falls nötig) zu speichern, damit man später alle Bytes beliebig oft analysieren kann. Danach kann man das LoRa Modul wieder zurück in den Empfangsmodus setzen. Das empfangene Paket kann nun interpretiert werden, dabei muss das Aufteilen und Kombinieren von Datenfeldern beim Senden wieder rückgängig gemacht werden. Bei mehreren Feldern in einem Byte schiebt man wieder alles in die andere Richtung um die gleiche Menge Bits zurück, bei aufgeteilten Feldern kombiniert man wieder alle Teile.

Spätestens jetzt beim Empfangen sollte man die Datenfelder auf der seriellen Konsole ausgeben. Dies geschah im vorherigen Beispielprogramm mit mehreren `print()` und `println()`, kann jedoch eleganter mit `sprintf()` durchgeführt werden. `sprintf(buffer, format ,value1, value2, ...)` ist ein starkes Werkzeug zum Formatieren von Text. Man gibt der Funktion einen ausreichend großen Puffer, einen Formatierungstext, beliebig viele Werte und schon hat man in seinem Puffer formatierten Text. Die Formatierung geschieht durch einen normalen Text mit Markierungen, an denen eine Zahl eingesetzt wird. `%d` nimmt eine Ganzzahl, `%f` eine Kommazahl, `%s` einen `char*` und so weiter. Auch können Zahlen z.B. auf eine bestimmte Menge an Nachkommastellen gekürzt oder auf eine Mindestlänge gestreckt werden. Eine komplette Erklärung der `printf()` Formatierungsmöglichkeiten wäre zu viel für diese Bastelhilfe. Wer mehr wissen will, kann sich Quelle (4) durchlesen.

Siehe Beispielprogramm 2 im Anhang zum Thema Daten Empfangen.

3 ANHANG

3.1 BEISPIELPROGRAMM 1: DATEN SENDEN

```
#include <LoRa.h>
#define LED_PIN 5
#define BUZ_PIN 6
#define MAGIC_NUM 0xAC

enum packet_types_e {
  PT_LED = 0,
  PT_TEXT = 1,
  PT_BEEP = 2,
  PT_NUMS = 3
};

void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  //Modul Starten und in den Bereitschaftsmodus setzten.
  LoRa.begin(866E6);
  LoRa.setSPIFrequency(1E6);
  LoRa.setTxPower(14);
  LoRa.idle();
  //Zufallsgenerator mit empfangenem Rauschen initialisieren.
  randomSeed(LoRa.random() * 100);
  delay(100);

  //Beginne das Paket
  LoRa.beginPacket();
  //Schreibe die "Magische" Zahl in das erste Byte,
  LoRa.write(MAGIC_NUM);
  //die Befehlsnummer in das nächste (zweite) byte,
  LoRa.write(PT_LED);
  //der Wert des analogen Einganges A0 geviertelt, sodass er in ein Byte passt,
  uint8_t led_bright = analogRead(A0) / 4;
  LoRa.write(led_bright);
  //und beende das Paket.
  LoRa.endPacket();

  //Gebe LED-Helligkeit auf der Seriellen Konsole aus
  Serial.print(F("LED[0-255]: "));
  Serial.println(led_bright);

  delay(100);

  LoRa.beginPacket();
  LoRa.write(MAGIC_NUM);
  LoRa.write(PT_TEXT);
  //Dieser Textteil kann eine beliebige Länge haben.
  LoRa.print("Das Pferd frisst keinen Gurkensalat");
  LoRa.endPacket();

  delay(100);
```

```
  LoRa.beginPacket();
  LoRa.write(MAGIC_NUM);
  LoRa.write(PT_BEEP);
  //Definiere <eine Variable des Typen 16bit unsignierter Integer> und <eine zweite als Array mit 2 Einträgen
des Typen byte> welche beide die gleichen Bytes enthalten.
  union {
    uint16_t freq_int;
    byte freq_bytes[2];
  };
  //Setzte die Frequenz auf einen Zufallswert zwischen 0 und dem max. eines uint16
  freq_int = random(0, 0xFFFF);
  //Schreibe von der 0ten Stelle von freq_b aus 2 bytes in das Paket
  LoRa.write(freq_bytes, 2);
  //Schreibe einen Zufallswert in das Tondauer-Byte
  uint8_t len_int = random(0, 0xFF);
  LoRa.write(len_int);
  LoRa.endPacket();

  //Toninfos auf Konsole ausgeben
  Serial.print(F("Tonfrequenz[Hertz]: "));
  Serial.println(freq_int);
  Serial.print(F("Tondauer[Centisekunden]: "));
  Serial.println(len_int);

  delay(100);


  LoRa.beginPacket();
  LoRa.write(MAGIC_NUM);
  LoRa.write(PT_NUMS);
  //Eine Zufällige Ganzzahl zwischen 0 und <<2 hoch 5> minus 1>, (die höchste in 5 Bits speicherbare Zahl)
wird gewählt
  uint8_t a_number_int = random(0, 31);
  //Bei der ersten Zahl wird sichergestellt, dass nur die rechten 5 bit genutzt werden können
  //???YYYYY -> 000YYYYY
  byte a_number = a_number_int & 0b00011111;
  //Eine Zufällige Ganzzahl zwischen 0 und <<2 hoch 3> minus 1>, (die höchste in 3 Bits speicherbare Zahl)
wird gewählt
  uint8_t another_num_int = random(0, 7);
  //Bei der 2ten wird sichergestellt das sie nur die rechten 3 bit nutzt. danach wird sie um 5 bit nach links
geschoben.
  //?????XXX -> XXX00000
  byte another_num = (another_num_int & 0b00000111) << 5;
  //Die bitweise ODER-Verknüpfung kombiniert beide
  //XXX00000 & 000YYYYY -> XXXYYYYY
  byte both_nums = a_number | another_num;
  LoRa.write(both_nums);
  LoRa.endPacket();

  //Zahlen auf Konsole ausgeben
  Serial.print(F("X: "));
  Serial.println(another_num_int);
  Serial.print(F("Y: "));
  Serial.println(a_number_int);
}

//Nichts mehr machen, wir sind fertig.
void loop() {}
```

3.2 BEISPIELPROGRAMM 2: DATEN EMPFANGEN

```
#include <LoRa.h>
#define LED_PIN 5
#define BUZ_PIN 6
#define MAGIC_NUM 0xAC

enum packet_types_e {
  PT_LED = 0,
  PT_TEXT = 1,
  PT_BEEP = 2,
  PT_NUMS = 3
};

void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZ_PIN, OUTPUT);
  //Modul Starten und in den Empfangsmodus setzten.
  LoRa.setSPIFrequency(1E6);
  LoRa.begin(866E6);
  LoRa.receive();
}

void loop() {
  //Abfragen ob ein Paket angekommen ist und wenn ja, wie lang es ist.
  uint8_t packetLength = LoRa.parsePacket();
  if (packetLength > 0) {
    //Array so groß wie Paket initialisieren.
    byte packet[packetLength];
    //LoRa Paket lesen, in Array speichern und auf der Konsole ausgeben.
    Serial.print(F("Paket empfangen: "));
    for (uint16_t b = 0; b < packetLength; b++) {
      packet[b] = LoRa.read();
      Serial.print(packet[b], HEX);
      Serial.print(' ');
    }
    Serial.println();

    //Modul wieder zurück in den Empfangsmodus setzen, da wir das paket gelesen haben
    LoRa.receive();

    //Wenn das erste Byte der "magischen" Zahl entspricht, nimm das 2te um zu entscheiden wie es weiter geht.
    if (packet[0] == MAGIC_NUM) switch (packet[1]) {
      case PT_LED:
        //setze LED helligkeit auf Wert von Byte 2
        analogWrite(LED_PIN, packet[2]);

        //Erzeuge einen bis zu 64 Zeichen langen Puffer
        char buf[64];
        //Formatriere den Infotext und schreibe ihn in den Puffer
        // %d setzt eine Ganzzahl ein, \r\n ist eine neue Zeile.
        sprintf(buf, "LED Helligkeit: %d\r\n", packet[2]);
        //Puffer auf Serieller Konsole ausgeben
        Serial.print(buf);
        break;

      case PT_TEXT:
        {
          //Gehe durch das Paket ab Byte 2 und spucke alles auf der seriellen Konsole aus.
          for (uint16_t b = 2; b < packetLength; b++) {
            Serial.write(packet[b]);
          }
          Serial.println();
        }
        break;
```

```
      case PT_BEEP:
        {
          //integer und byte[2] enthalten die gleichen Bytes
          union {
            uint16_t freq_i;
            byte freq_b[2];
          };
          //Kopiere nach Stelle 0 in freq_b, von packet ab Stelle 2, 2 Bytes
          memcpy(&freq_b[0], &packet[2], 2);

          //Spiele einen Ton von Frequenz (Byte 2 und 3) Hertz für (Byte 4 mal 10) Millisekunden
          tone(BUZ_PIN, freq_i, packet[4] * 10);

          //Infos über ton auf Konsole ausgeben.
          char buf[64];
          sprintf(buf, "Ton: %dHertz, %dMillisekunden\r\n", freq_i, packet[4] * 10);
          Serial.print(buf);
        }
        break;

      case PT_NUMS:
        {
          //Nur die rechten 5 Bits werden gespeichert
          //XXXYYYYY -> 000YYYYY
          uint8_t a_number = packet[2] & 0b00011111;
          //die 3 linken Bits werden um 5 nach rechts geschoben, freier Raum wird mit 0 gefüllt
          //XXXYYYYY -> 00000XXX
          uint8_t another_num = packet[2] >> 5;

          //Zahlen auf Konsole ausgeben.
          char buf[64];
          sprintf(buf, "X: %d, Y: %d\r\n", another_num, a_number);
          Serial.print(buf);
        }
        break;
      }
  }
}
```

3.3 FREQUENZBÄNDER

- 433,05MHz-434,79MHz
  - Maximal 10dBm/10mW Sendeleistung
  - Keine Sendezeitsbeschränkung
- 863MHz – 865MHz
  - Maximal 14dBm/25mW Strahlungsleistung (ERP)
  - Maximal 3,6s! (0,1%) aktiv am Senden, innerhalb 60 Minuten
- 865MHz – 868MHz
  - Maximal 14dBm/25mW Strahlungsleistung (ERP)
  - Maximal 36s (1%) aktiv am Senden, innerhalb 60 Minuten
- 869,40MHz – 869,65MHz
  - Maximal 27dBm/500mW Strahlungsleistung (ERP)
  - Maximal 6min (10%) aktiv am Senden, innerhalb 60 Minuten

Quelle: (5)

Dies ist kein rechtlicher Rat. Keine Gewähr auf jegliche Angaben im gesamten Dokument. Ich hafte nicht wenn die Leute im Peilwagen euch heimsuchen kommen.

4 LITERATURVERZEICHNIS

1. **Semtech.** LoRa Modulation Basics. *Semtech.* [Online] 2015. [Zitat vom: 11. 01 2023.] https://semtech.my.salesforce.com/sfc/p/E0000000JelG/a/2R0000001OJa/2BF2MTeiqIwkmxkcjjDZzalPUGlJ76lLdqiv.30prH8.

2. —. SX1276-7-8-9 Datasheet. *Mouser.* [Online] 2016. [Zitat vom: 29. 12 2022.] https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf.

3. **sandeepmistry + contributors.** LoRa API. *arduino-LoRa Repository on GitHub.* [Online] 02 22, 2021. [Cited: 01 11, 2023.] https://github.com/sandeepmistry/arduino-LoRa/blob/master/API.md.

4. **cplusplus.com.** printf. *cplusplus.com.* [Online] [Zitat vom: 13. 01 2023.] https://legacy.cplusplus.com/reference/cstdio/printf/.

5. **Bundesnetzagentur.** Funkanwendungen mit geringer Reichweite; Non-specific Short Range Devices (SRD). *Bundesnetzagentur.* [Online] 18. 12 2014. [Zitat vom: 29. 12 2022.] https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/Allgemeinzuteilungen/FunkanlagenGeringerReichweite/2018_05_SRD_pdf.pdf?__blob=publicationFile&v=7.

Lest euch gerne Quelle (1) durch wenn ihr mehr darüber wissen wollt wie LoRa genau funktioniert.