**LCD MAKERSPACE: Radiowecker**

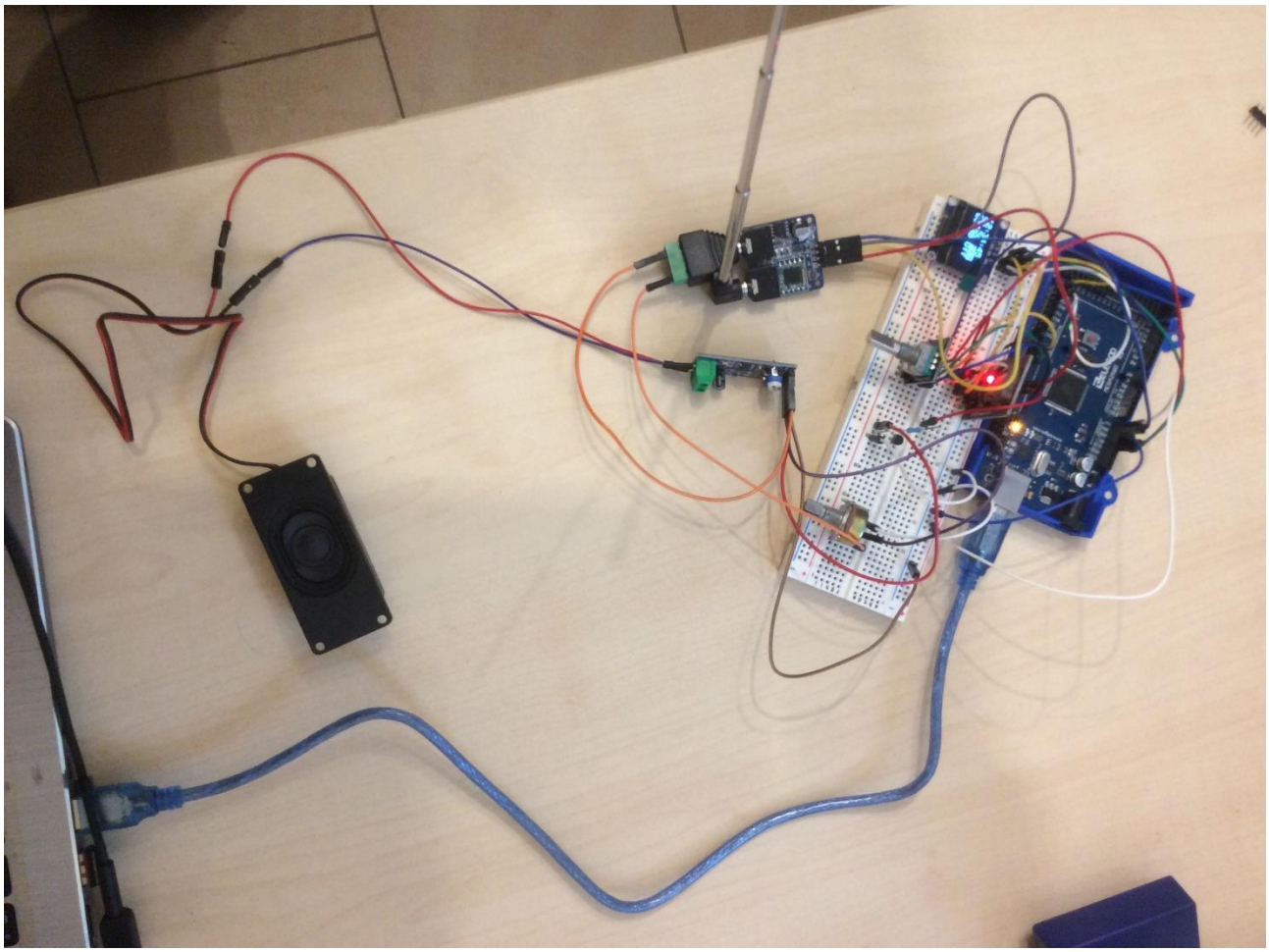
Dieses Dokument zeigt, wie man selbst einen Radiowecker bauen kann. Nach und nach werden die verschiedenen Bauteile vorgestellt und verbunden.

Versuche alle Schritte einzuhalten. Wenn dir das gesamte Projekt jedoch irgendwann zu komplex erscheint, kannst du natürlich auch Anpassungen vornehmen.

Du könntest beispielsweise „nur“

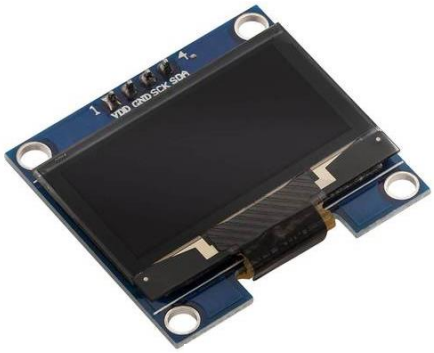
- eine Uhr mit Bildschirm oder
- ein Radio ohne Bildschirm
- ein tragbares Radio das mit Kopfhörern funktioniert

bauen. Du könntest auch eine Bedienung mit Fernbedienung oder Schaltern oder einem Joystick wählen.



# 1. Der OLED Bildschirm

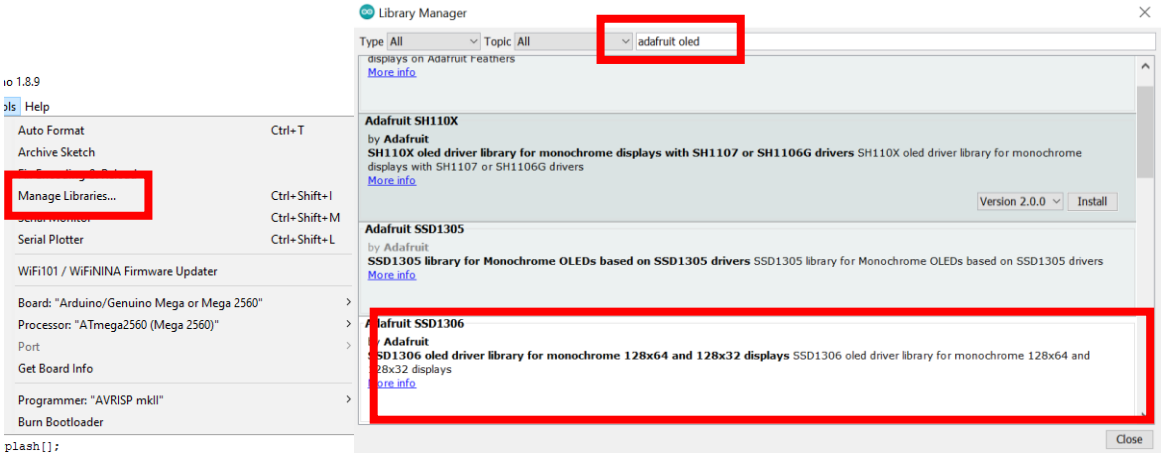
Große OLED-Bildschirme können in TV-Geräten verbaut oder als PC-Bildschirm genutzt werden. In Arduino-Projekten kann man Informationen (Zeit, Temperatur, beliebigen Text), kleine Diagramme zeichnen und Menüs, die der Steuerung von Geräten dienen, anzeigen. Später werden wir den Bildschirm benutzen, um z.B. die aktuelle Uhrzeit, die gewünschte Alarmzeit und die Frequenz des Radios anzuzeigen.



Stecke den Bildschirm auf das Breadboard. Verbinde den Bildschirm wie folgt mit dem Arduino. Eventuell musst du vorher eine Stift-Leiste anlöten.

| Pin Bildschirm | Pin Arduino |
|---|---|
| GND | GND |
| VCC oder + | 3.3V oder 5V |
| SCL | SCL |
| SDA | SDA |

Installiere folgende Bibliothek (und die automatisch vorgeschlagenen) für den Bildschirm




Schreibe das Programm ab und lies die Kommentare (hellgrau, hinter // oder /*).

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>


#define SCREEN_WIDTH 128 // OLED Anzahl Pixel horizontal (Kolonnen)
#define SCREEN_HEIGHT 64 // OLED Anzahl Pixel vertikal (Zeilen)

//Bildschirm und Anschlussart definieren
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup() {
 Serial.begin(115200);

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { //Bildschirm starten
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
```

```
  delay(2000);
  display.clearDisplay();         //Bildschirm leeren

  display.setTextSize(2);         //Schriftgröße (der ganze Bildschirm hat eine Höhe von 8)
  display.setTextColor(WHITE);    //WHITE wenn der Hintergrund dunkel ist, sonst BLACK
  display.setCursor(0, 0);       //  Wo (Zeile, Kolonne) soll geschrieben werden?
                                  // WICHTIG: DIe erste Zeile, Kolonne hat die Nummer 0!!
                                  // Wir beginnen ganz oben links
```

Experimentiere selbst mit den Funktionen, bis du verstehst, wie der Bildschirm sich verhält.

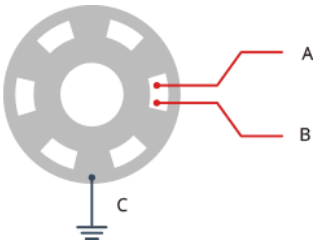
```
  // Was (das Wort "Test") soll geschrieben werden
  // Wenn wir println benutzen, wird automatisch die Zeile gewechselt(bei print nicht)
  display.println("Test");

  //  Das Wort "Test2" soll  in die nächste Zeile geschrieben werden
 // display.setCursor(1, 3);
  display.println("Test2");

  //  Das Zahl 8751 soll  in die nächste Zeile geschrieben werden
  display.println(8751);

display.setTextColor(BLACK, WHITE); //"negative" Schrift
  display.println("negativ");

  //vorher wurde nur angegeben was wo geschrieben werden soll.
  //Erst die nächste Zeile erneuert den Bildschirminhalt
  display.display();
```

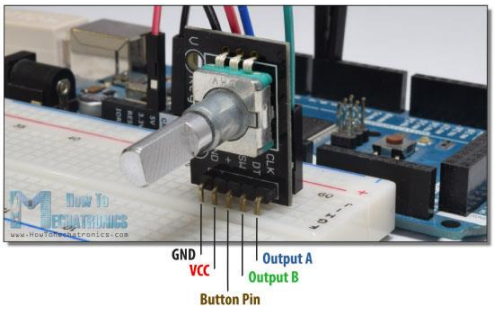
Der nächste Teil des Programms (kann direkt daruntergeschrieben werden), zeigt einige fortgeschrittene Funktionen. Ändere auch hier einige Optionen, bis du verstehst, was die Befehle bedeuten.

```
  // Wir warten 5 Sekunden, bereiten neue Anweisungen vor und aktualisieren den Bildschirm
  delay(6000);
  display.clearDisplay();      //Blidschirm leeren
  display.setCursor(0,0);
  display.println("Die dritte");
  display.setTextColor(WHITE,BLACK ); //Farben wieder normal
  display.println("Zeile");
  display.println("wandert");
  display.display();
  display.startscrollright(0x04, 0x05);
// Der Bildschirm ist in 8 Teile 0x00 bis 0x07 aufgeteilt
// Wir wählen, welche sich bewegen (dies hängt von der Schriftgröße ab)

}

// the loop routine runs over and over again forever:
void loop() {


}
```

## 2. Der Drehregler (Rotary Motion Encoder)

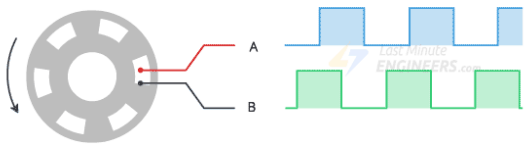
Wenn du das Roboterauto gebaut hast, weißt du, wie man eine Fernbedienung nutzen kann. Hier wirst du lernen, wie man einen Drehregler nutzen kann, um ein Gerät zu steueren.



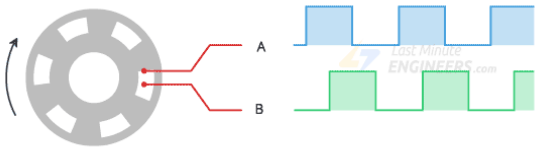




Es handelt sich um ein Rad mit Öffnungen. Wenn wir am Rad drehen, werden die Kontakte A und B (normalerweise 5V also Signal = 1) nacheinander mit GND verbunden (Signal=0)

Prinzip: Wenn A ändert, wird am Knopf gedreht. Wenn $A = 1$ und $B = A$, dann wurde nach links gedrekt. Wenn $A = 1$ und $B \neq A$ wurde nach rechts gedreht.




Counter Clockwise Rotation




Clockwise Rotation

In unserem Programm wird es eine Variable „counter“ geben, die anfangs 0 ist. Sie wird größer, wenn wir nach rechts drehen, und kleiner, wenn wir nach links drehen.

Der Regler selbst kann auch eingedrückt werden und funktionniert wie ein Schalter. Der Button-Pin erhält das Signal 0, wenn der Knopf gedrückt wird. So können wir erkennen, bei welcher Position des Reglers, der Knopf gedrückt wurde.

Trenne den Bildschirm von Arduino-Modul und schließe den Drehregler wie folgt an.

| Pin Drehregler | Pin Arduino |
|---|---|
| GND | GND |
| VCC oder + | 5V |
| SW | Pin 2 |
| DT | Pin 4 |
| CLK | Pin 3 |

Tippe das Programm ab und lies die Kommentare. Das Programm wird nichts steuern aber am Serial Monitor anzeigen, wie der Regler ausgelesen wird.

```
const int CLK = 3;
const int  DT = 4;
const int SW = 2;

int counter = 0;                    //Diese Zahl wird größer oder kleiner, wenn wir den Knopf drehen
int currentStateCLK;                // CLK ändert sich wenn der Knopf  gedreht wird.
int lastStateCLK;                   //Wir speichern den neuen Wert und den alten Wert
char currentDir =' ';              // Hier wird "R" oder "L" gespeichert, je nachdem wie der Knopf gedreht wird
int btnState=0;                     //LOW wenn der Knopf gedrückt wurde, sonst HIGH
unsigned long lastButtonPress = 0;  //Wann wurde der Knopf gedrückt? (in ms)


void setup() {

  // Pins für den Drehregler definieren
  pinMode(CLK,INPUT);
  pinMode(DT,INPUT);
  pinMode(SW, INPUT_PULLUP); //WICHTIG, sonst müsste man nach einen Widerstand verbauen
```

```
  // Setup Serial Monitor
  Serial.begin(115200);

  // Read the initial state of CLK
  lastStateCLK = digitalRead(CLK); // ersten Wert von CLK lesen
}

void loop() {

  // aktuellen Wert von CLK lesen
  currentStateCLK = digitalRead(CLK);

  // Wenn der Zustand von CLK ändert, wurde am Knopf gedreht
  // Die Bedingung hinter && sorgt dafür, dass sauber ausgelesen wird
  if (currentStateCLK != lastStateCLK  && currentStateCLK == 1){

    // Wenn DT und CLK verschieden sind: nach links gedreht, der Wert des Zählers wird kleiner
    if (digitalRead(DT) != currentStateCLK) {
      counter --;
      currentDir ='L';
    } else {
      // Ansonsten: es wurde nach rechts gedreht, der Wert des Zählers wird kleiner
      counter ++;
      currentDir ='R';
    }


    Serial.print("Richtung: ");
    Serial.print(currentDir);
    Serial.print(" | Counter: ");
    Serial.println(counter);
  }

  // den CLK speicheren, damit er im nächsten Zyklus zum Vergleich genutzt werden kann
  lastStateCLK = currentStateCLK;

  // Lesen, ob der Knopf gedrückt wurde
  int btnState = digitalRead(SW);

  //Bei LOW wurde der Knopf gerückt
  if (btnState == LOW) {
    //wir gehen davon aus, dass mindestens 50 ms zwischen 2 Betätigungen liegen
    if (millis() - lastButtonPress > 50) {
      Serial.print("Knopf wurde gedrückt bei  ");
            Serial.println(counter);
    }

    // Remember last button press event
    lastButtonPress = millis();
  }

  // Kurz warten
  delay(1);
}
```

## 3. Interrupts

Das Programm vorhin funktioniert. Aber, die ganze Loop-Sektion wird durch das Auslesen des Drehreglers und des Knopfs blockiert. Später wollen wir aber, dass das Programm weitere Befehle ausführt. Der CLK-Pin des Drehreglers und der Schalter-Pin sollen überwacht werden aber das Programm soll sie ignorieren, bis etwas passiert. Hier kommen die Interrupts in Spiel. Sie unterbrechen (interrompre, to interrupt) das Programm, wenn es zu einer Änderung an den überwachten Pins kommt.

Das neue Programm ist dem alten sehr ähnlich, allerdings gibt es wichtige Unterschiede.

- Alle Variablen, die vom Drehregler abhängen müssen als *volatile* definiert werden

```
const int CLK = 3;
const int  DT = 4;
const int SW = 2;

volatile int counter = 0;                    //Diese Zahl wird größer oder kleiner, wenn wir den Knopf drehen
volatile int currentStateCLK;                // CLK ändert sich von 1 auf 0, wenn er Knopf  gedreht wird.
volatile int lastStateCLK;                   //Wir speichern den neuen Wert und den alten Wert
volatile char currentDir =' ';              // Hier wird "R" oder "L" gespeichert, je nachdem wie der Knopf gedreht wird
unsigned long lastButtonPress = 0;


volatile boolean ButtonDetected;      //Hier lesen wir, ob der Knopf gedrückt wurde
```

- Im Setup-Teil teilen wir dem Programm mit, welche Pins überwacht werden sollen.

```
void setup() {

  // Pins für den Drehregler definieren
  pinMode(CLK,INPUT);
  pinMode(DT,INPUT);
  pinMode(SW, INPUT_PULLUP);

  // Setup Serial Monitor
  Serial.begin(115200);

  // Read the initial state of CLK
  lastStateCLK = digitalRead(CLK); // ersten Wert von CLK lesen
   attachInterrupt(digitalPinToInterrupt(CLK), readRotary, CHANGE);
   //Wir rufen die Funktion readRotary auf, wenn am Knopf gedreht wird, also wenn CLK ändert
   attachInterrupt(digitalPinToInterrupt(SW), buttonDetect, FALLING);
   //Wir rufen die Funktion buttonDetect auf, wenn der Knopf gedrückt wird, also wenn SW kleiner wird
}
```

- Es gibt nun einige Funktionen, nämlich die, die ausgeführt werden, wenn die Interrupts aktiviert werden. Schriebe sie zwischen *setup* und *loop*.

  1. Wir teilen dem Rest des Programms, dass der Knopf gedrückt wurde, indem wir eine Variable ändern.

```
void buttonDetect ()  {
  ButtonDetected = true;
}
```

  2. Rotationsrichtung auslesen und den Counter anpassen. Also quasi der Hauptteil des vorigen Programms

```
void readRotary(){

  currentStateCLK = digitalRead(CLK);

  if (currentStateCLK != lastStateCLK  && currentStateCLK == 1){

    if (digitalRead(DT) != currentStateCLK) {
      counter --;
      currentDir ='L';
    } else {

      counter ++;
      currentDir ='R';
    }

    Serial.print("Richtung: ");
    Serial.print(currentDir);
    Serial.print(" | Counter: ");
    Serial.println(counter);
  }


  lastStateCLK = currentStateCLK;


}
```

3. Was passiert, wenn der Knop gedrückt wurde?

```
void doButton(){

    Serial.print("Knopf wurde gedrückt bei  ");
          Serial.println(counter);

  }
```

4. In der Loop-Sektion steht fast nichts mehr
   Wir wollen hier nur dafür sorgen, dass der Knopf nicht zu oft ausgelesen wird.

```
void loop() {

        if (ButtonDetected)  {

    if (millis() - lastButtonPress > 500)doButton(); //Der Knopf darf nicht zu oft ausgelesen werden
    lastButtonPress = millis();
    ButtonDetected = false;


      }
  }
```

## 4. Die Uhr

Das Arduino-Board verfügt auch über eine Uhr, aber sie funktioniert nur, wenn das Board mit Strom versorgt wird. Deshalb benutzten wir eine Uhr (z.B. DS1307), welche mit Hilfe einer kleinen Batterie weiter funktioniert, wenn das Arduino-Board ohne Versorgung ist.

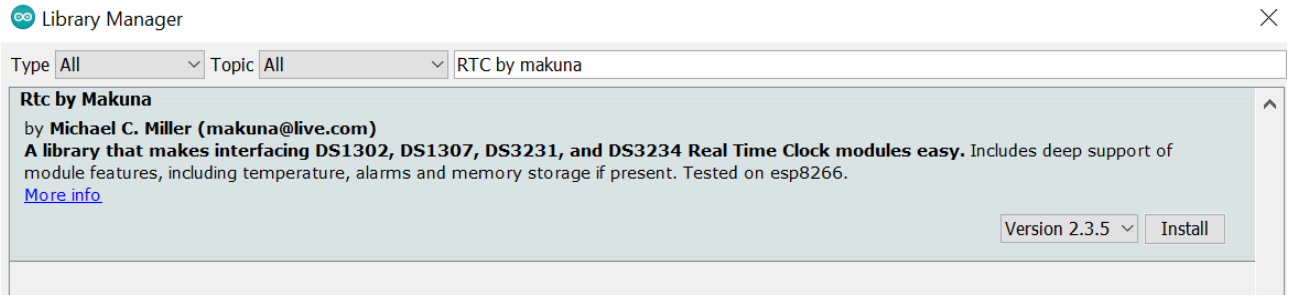
Mit diesem Programm wollen wir die aktuelle Uhrzeit auf dem OLED Display anzeigen.

Trenne den Drehregler vom Arduino-Board und schließe den Bildschirm an.

Stecke die Uhr auf das Breadboard und verbinde sie mit dem Arduino.

| Pin Uhr | Pin Arduino |
|---|---|
| GND | GND |
| VCC oder + | 5V |
| SCL | SCL |
| SDA | SDA |

Installiere folgende Bibliothek.




Öffne das erste Programm (Bildschirm) und speichere es unter einem neunen Namen. Vieles kannst du nämlich wiederverwenden. In den Programmkommentaren werden die neuen Funktionen erklärt.

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>


#include <RtcDS1307.h> //Bibliothek für Uhr
//https://github.com/Makuna/Rtc/wiki/RtcDS1307-object


#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);


//Uhrsensor und Anschlussart definieren
RtcDS1307<TwoWire> Rtc(Wire);
RtcDateTime now; //die aktuelle Zeit wird hier gespeichert
char zeit[9]="";//die aktuelle Zeit wird hier als Text gespeichert
```

```
void setup() {
    Serial.begin(115200);
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
      Serial.println(F("SSD1306 allocation failed"));
      for(;;);
    }


    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);


//Uhr initialisieren
    Rtc.Begin();


// Hier wird die Computerzeit mit der Zeit auf dem DS1307 verglichen.
//Danach wird die Uhr gestellt
    RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);

    if (!Rtc.GetIsRunning())
    {
        Serial.println("RTC was not actively running, starting now");
        Rtc.SetIsRunning(true);
    }

    now=Rtc.GetDateTime();



    RtcDateTime now = Rtc.GetDateTime();
    if (now < compiled) Rtc.SetDateTime(compiled);
    Rtc.SetSquareWavePin(DS1307SquareWaveOut_Low);
}

 //Zeit auslesen und Bildschirm aktualisieren
 void updatedisplay(){

     display.setCursor(0, 0);

   //Wir erzeugen einen Text mit der aktuellen Zeit (jeweils 2 Stellen)
     now=Rtc.GetDateTime();
     Serial.println(now.Second());
     sprintf(zeit, "%02d:%02d:%02d", now.Hour(), now.Minute(), now.Second());

     display.clearDisplay();
     display.println(zeit);
     display.display();
     Serial.println(zeit);
 }




 void loop() {
 //Jede Sekunde wird die Uhr aktualisiert
     updatedisplay();
     delay(1000);
 }
```

## 5. Zeitschaltuhr (Minuterie)

Am Ende bauen wir einen Radiowecker. Wir können aber jetzt schon Geräte mit Hilfe des Gelernten zu einer bestimmten Zeit einschalten. Hier werden wir eine LED steuern und alles ist bereits vorbereitet, um später das Radio anzusteuern.

Wegen der verschiedenen Bedienungselemente ist das Hauptprogramm ziemlich komplex. Lade es am besten von der Teams-Seite herunter. Du müsstest die meisten Teil wiedererkennen (Bildschirm, Drehregler, Uhr). Allerdings wurden etliche neue Variablen eingefügt und die Funktion *doButton* wurde wesentlich komplizierter. Die Funktion *updatedisplay* zeigt regelmäßig alle Informationen an.

Du sollst hier das Programm testen und verstehen, wie man es benutzt.

Verbinde Pin8 mit dem +-Anschluss(langes Bein) einer LED. Das andere Bei mit einem $220\Omega$ Widerstand und das freie Ende des Widerstands mit GND:

Was kann das Programm?

- Es zeigt die aktuelle Uhrzeit an
- Wenn du den Drehregler drehst, schaltest du zwischen verschiedenen Menüpunkten
    - Die erste Zahl ist die Frequenz.
        - Drücke auf den Knopf.
        - Drehe am Regler und wähle eine neue Frequenz
        - Drücke auf den Knopf, um sie zu bestätigen
        - Ein diesem Programm mit der LED sieht man nur eine andere Zahl, es passiert aber sonst nichts.
    - Rad
        - Drücke auf den Knopf um das Radio ein- oder auszuschalten.
        - Hier müsste die LED an- oder ausgehen.
    - Alarmzeit
        - Ähnlich wie bei der Frequenz kannst du hier die Alarmzeit (Stunden und Minuten getrennt) einstellen.
    - On: Alarm erwünscht Off: kein Alarm erwünscht.
- Schalte die LED aus. Stelle eine Alarmzeit ein, die 1-2 Minuten in der Zukunft liegt. Sorge dafür dass „On“ erscheint und warte.
- Schalte die LED anschließend aus.

## 6. Das Radiomodul

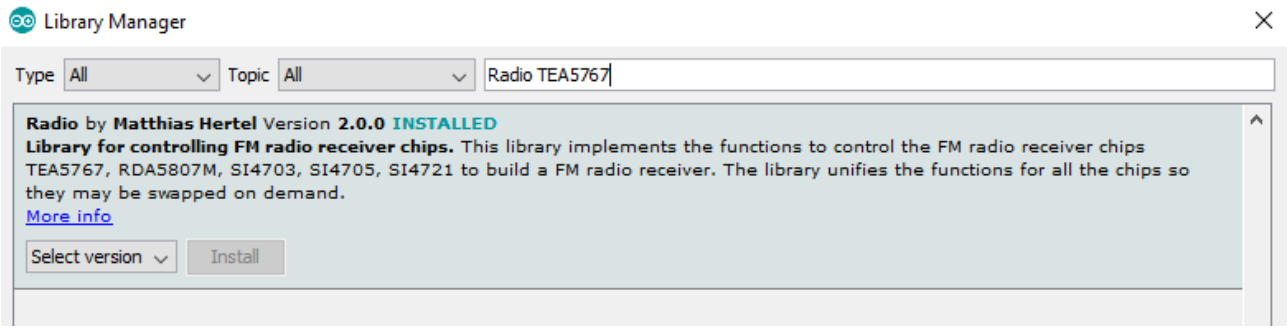
In diesem Teil werden wir das Radiomodul alleine testen. Trenne das Arduino-Board vom Breadboard. **Lass aber den ganzen Rest aufgebaut!**

Wir benutzen das Modul TEA5767. Der kleine grüne Chip ist der eigentliche Radioempfänger. Der Rest ist ein kleiner Verstärker, der dafür sorgt, dass man das gewählte Programm mit einem (3,5mm Klinke) Kopfhörer hören kann.

Oben wird die Antenne angeschlossen, unten der Kopfhörer.

| Pin Radiomodul | Pin Arduino |
|---|---|
| GND | GND |
| VCC oder + | Hier: 5V<br>Später werden wie hier den Pin8 verbinden. Anstelle der LED können wir so das Radio ein- oder ausschalten. |
| SCL | SCL |
| SDA | SDA |

Installiere folgende Bibliothek. Sie sorgt dafür, dass das Radio sehr einfach gesteuert werden kann.



Schreibe das folgende Programm ab. In der Variable *frequency* wird die Frequenz gespeichert.

RTL Luxemburg hat die Frequenz 92,5MHz oder 88,9 MHz. Dem Programm müssen wir in dem Fall 9250 und 8890 angeben.

```
#include <radio.h>
#include <TEA5767.h>

TEA5767 radio;

int frequency=9250;

void setup() {



   radio.init();
    Serial.println("init");
  radio.setMono(false);
  radio.setBandFrequency(RADIO_BAND_FM, frequency);
  delay(2000);
  radio.setVolume(1);
```

Teste das Programm und versuche Eldoradio (95MHz) zu hören.

## 7. Das (fast) fertige Programm

- Schließe die vorigen Bauteile wieder an.
- Die LED und der Widerstand werden nicht mehr gebraucht.
- Verbinde Pin8 des Arduino mit VCC des Radiomoduls
- Lade das Programm herunter und teste es. Anstelle der LED steuerst du nun das Radio. Die Frequenz kannst du während des Betriebs regeln

## 8. Laustsprecher

Die aktuelle Version hat einige Nachteile

- man kann nur einen Kopfhörer anschließen
- die Lautstärke lässt sich nicht ändern

Wir müssen also einen Verstärker anschließen.

- Trenne die Stromversorgung des Arduino
- Stecke einen Adapter in die Kopfhörerbuchse.
- Klemme jeweils ein Kabel in L und GND (das Symbol das weder L noch R ist).

- Schließe den Verstärker wie folgt an

| Pin Verstärker links | |
|---|---|
| VCC | VIN des Arduino |
| IN | L aus dem Adapter |
| GND | GND aus dem Adapter |
| GND | GND des Arduino |

| Pin Verstärker rechts (grüne Klemmen) | |
|---|---|
| OUT | + (rotes Kabel) des Lautsprechers |
| GND | - (blaues Kabel) des Lautsprechers |

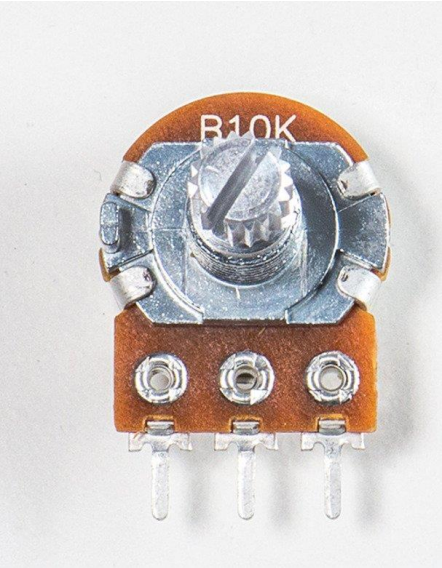
Schließe das Arduino wieder an. Der Lautsprecher müsste jetzt funktionieren. Die Lautstärke kanns du mit Hilfe eines Schraubenziehers regeln indem du die Stellung des Potentiometers (blaues Teil) änderst.

## 9. Lautstärke selbst regeln

Es ist deutlich praktischer, die Lautstärke mit einem eigenen Potentiometer zu regeln. Wähle eines von $10\ k\Omega$ oder $20\ k\Omega$.

Schalte das Arduino aus

Stecke das Potentiometer aus des Breadboard und schließe es wie folgt an:




Links: GND des Arduino

Mitte: IN des LM386. Du musst also das Kabel L (Adapter) lösen und umstecken)

Rechts: L aus dem Adapter

Schalte das Arduino ein.

Drehe am Potentiometer. Die Lautstärke sollte sich ändern.

Wähle die Position des blauen Potentiometers(LM386) so, dass im ganzen Bereich des neuen Potentiometer ein unverzerrtes Signal und nicht übertrieben lautes Signal hörbar ist.

## 10.Transistor

Ein letztes Problem gibt es. Wenn das Radio ausgeschaltet ist, hört man ein Rauschen. Dies liegt daran, dass der Verstärker noch immer mit Strom versorgt wird und Störsignale empfängt und verstärkt.

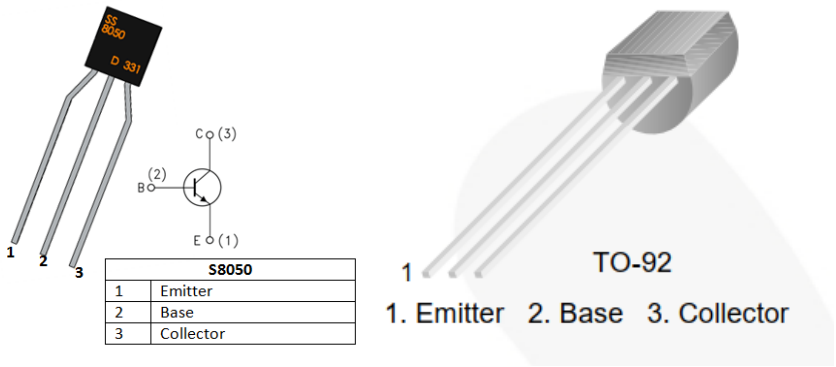
Da der Verstärker jedoch einen ziemlich großen Strom benötigt, würden wir das Arduino beschädigen, wenn wir den Verstärker mit Hilfe eines Pins versorgen. Wir benutzen daher einen Transistor.

### Teil 1: Mehrere LEDs mit 1 PIN ansteuern

Ein Arduino-Ausgangspin kann nur einen kleinen Strom liefern. Trotzdem soll der kleine Strom mehrere LEDs, einen Motor oder den Radioverstärker ansteuern (hoher Strom).

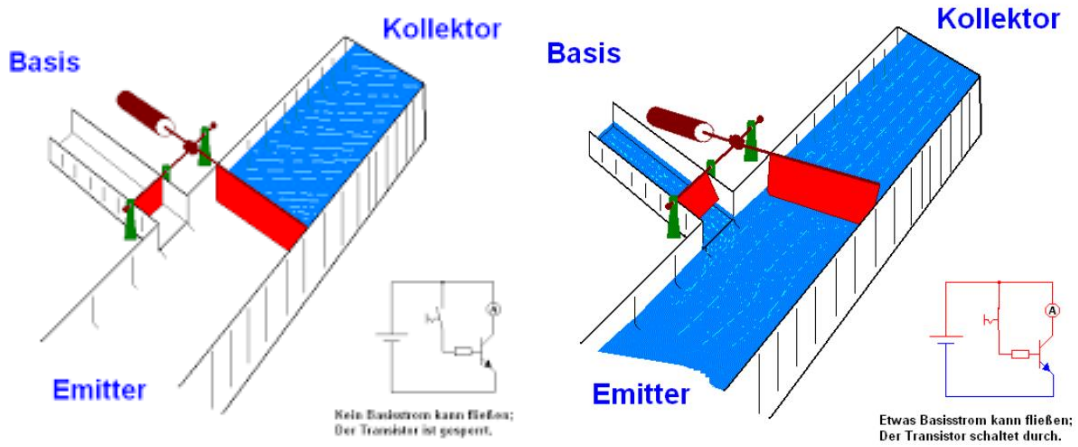
Benutze am besten ein anderes Arduino, ein leeres Breadboard. Du brauchst außerdem 3-5 LEDS, 3-5 Widerstände von 220Ω und einen S8050 Transistor.

Ein Transistor hat 3 Anschlüsse, die man auf keinen Fall verwechseln darf (Achte auf die Rundung!): Kollektor C, Basis B, Emitter E.

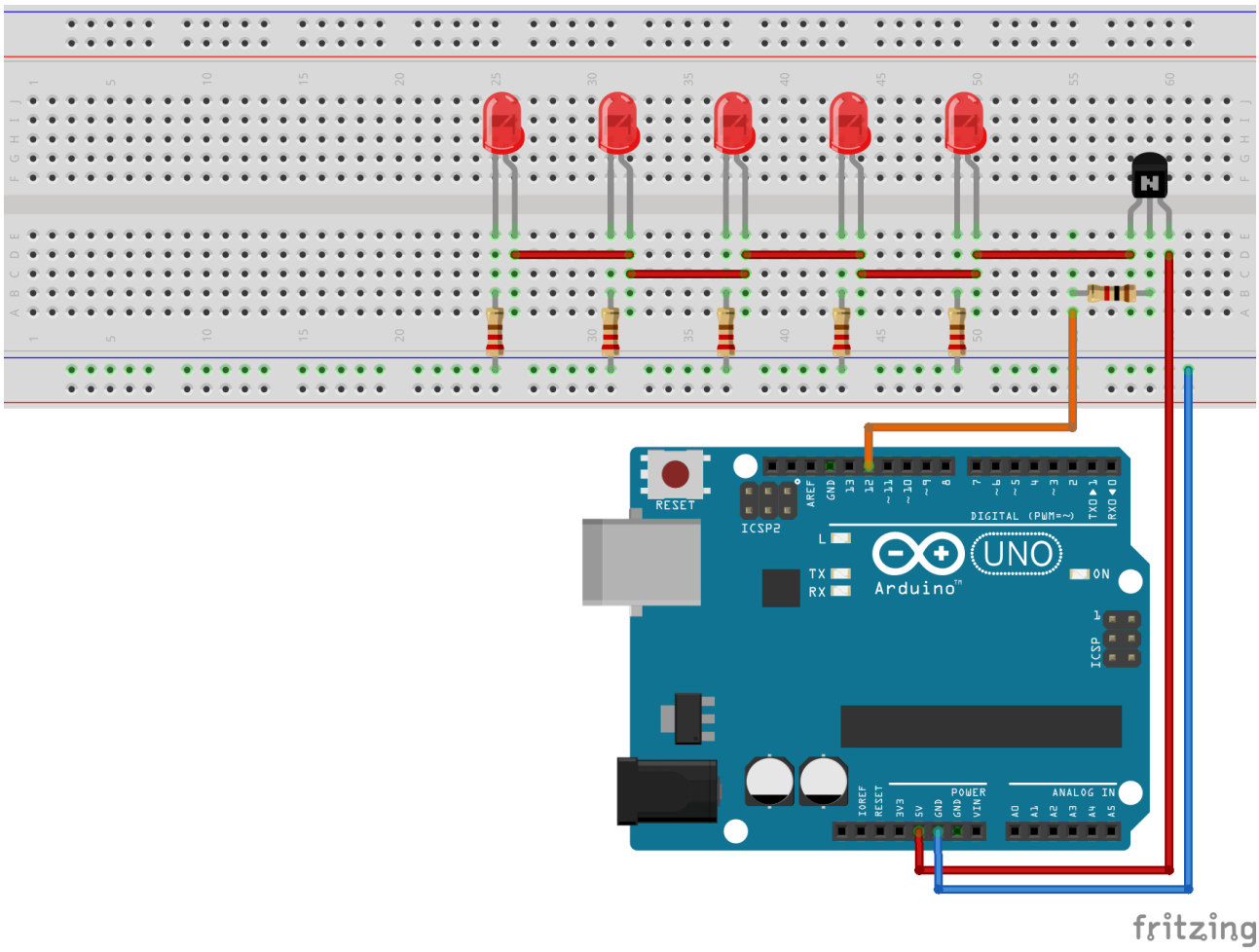



| S8050 | |
|---|---|
| 1 | Emitter |
| 2 | Base |
| 3 | Collector |

Prinzip: Ein kleiner Strom in die Basis (z.B. von einem Arduino-Pin) kann dafür sorgen, dass ein großer Strom zwischen Kollektor und Emitter fließt.



Baue den folgenden Stromkreis. 3 LEDS und Widerstände reichen aus.




Der Programmcode ist sehr einfach:

```
int BasisPin=12;

void setup(){
  pinMode(BasisPin, OUTPUT);      // Den Basispin als Ausgang festlegen.
  digitalWrite(BasisPin, LOW);
}

void loop(){
  digitalWrite(BasisPin, HIGH);  // Strom duch die Basis fließen lassen
  delay(1000);
  digitalWrite(BasisPin, LOW);     // Basisstrom ausschalten
  delay(1000);
}
```

Wenn der Pin 12 einen kleinen Strom zur Basis des Transistors liefert, das fließt ein großer Strom vom Kollektor zum Emitter, also auch durch die LEDs.

Die 5V- und GND-Anschlüsse können ohne weiteres den benötigten Strom liefern. Würde man auf den Transistor verzichten und den Pin 12 direkt an alle LEDs anschließen, wäre der Pin 12 zerstört.

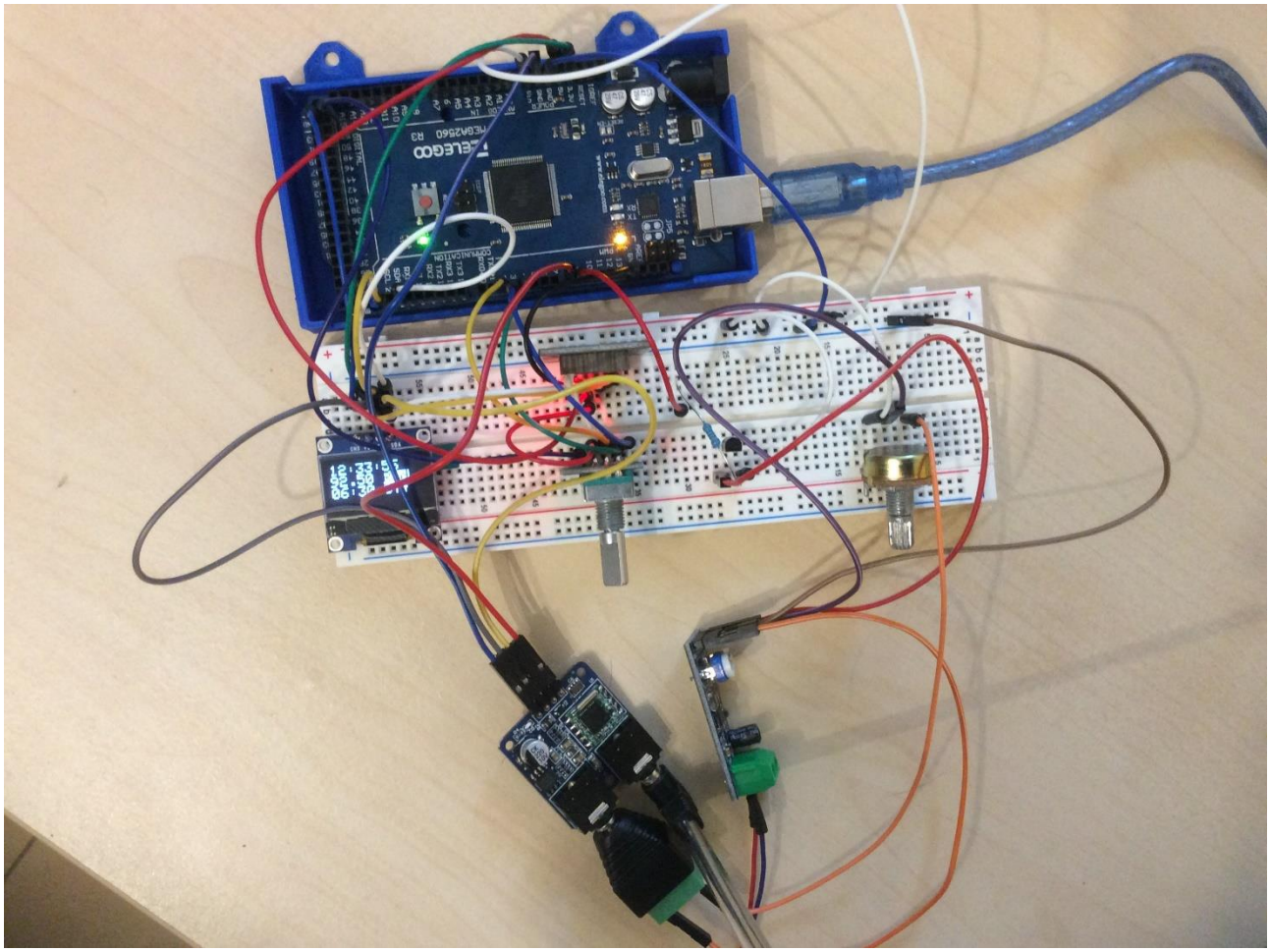
Bei noch größeren Strömen (siehe Motor Roboterauto) muss man eine externe Stromversorgung benutzen.

**Teil2: Das fertige Radio mit Verstärker**

Führe folgende Änderungen am Radiostromkreis durch.

- **Stecke den Transistor wie in Teil 1 ins Breadboard**
- **Rechts ist der Kollektor. Verbinde ihn mit Vin des Arduino-Boards**
- **In der Mitte ist die Basis. Verbinde sie mit einem $1k\Omega$-Widerstand. Das freie Ende des Widerstands verbindest du mit Pin 9 des Arduino.**
- **Links ist der Emitter. Verbinde ihn mit VCC des LM386**

Der Aufbau ist also derselbe wir in Teil 1, nur dass wir nicht die erste LED sondern den VCC-Anschluss des Verstärkers mit dem Emitter verbinden.



Lade die letzte Version des Programms herunter. Es gibt minimale Ergänzungen:

```
const int transPin=9;
```

In setup definieren wir einen weiteren Pin.

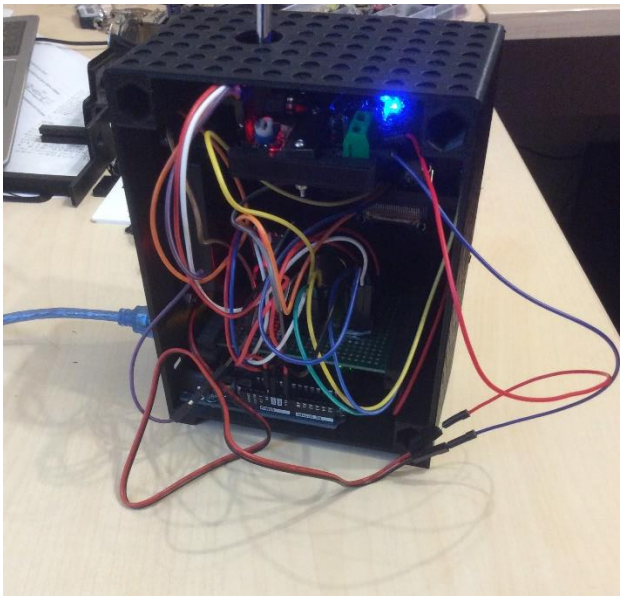
```
pinMode(transPin, OUTPUT);
digitalWrite(transPin,LOW);
```

Wenn wir das Radio ein- oder ausschalten, wird zusätzlich der Verstärker mit Hilfe des Transistors geschaltet.

```
void startradio(){
   digitalWrite(radioPin,HIGH);
  delay(1000);
   radio.init();
  radio.setMono(false);
  radio.setBandFrequency(RADIO_BAND_FM, 10*frequency);
   radio.setVolume(1);
   digitalWrite(transPin,HIGH);
}


void stopradio(){
   digitalWrite(radioPin,LOW);
     digitalWrite(transPin,LOW);
}
```

## 11.Abschließende Bemerkungen

Natürlich ist das momentane Radio unpraktisch in der Handhabung. Man kann den Stromkreis auf Platinen verlöten und das Ganze in ein Gehäuse (3D-Drucker!) setzen.



Ein gewisses Rauschen ist möglich. Je mehr Bauteile miteinander werden, desto mehr Signalstörungen können auftreten. Lange Kabel funktionieren wie Antennen und können zusätzliche Signale auffangen. Das Herausfiltern der Störungen ist sehr schwierig und wird deshalb hier nicht behandelt.

Je einfacher das Projekt ist, desto besser ist die Tonqualität. Wenn dich die anderen Funktionen nicht interessieren, kannst du dich beispielsweise auf das Radio beschränken.