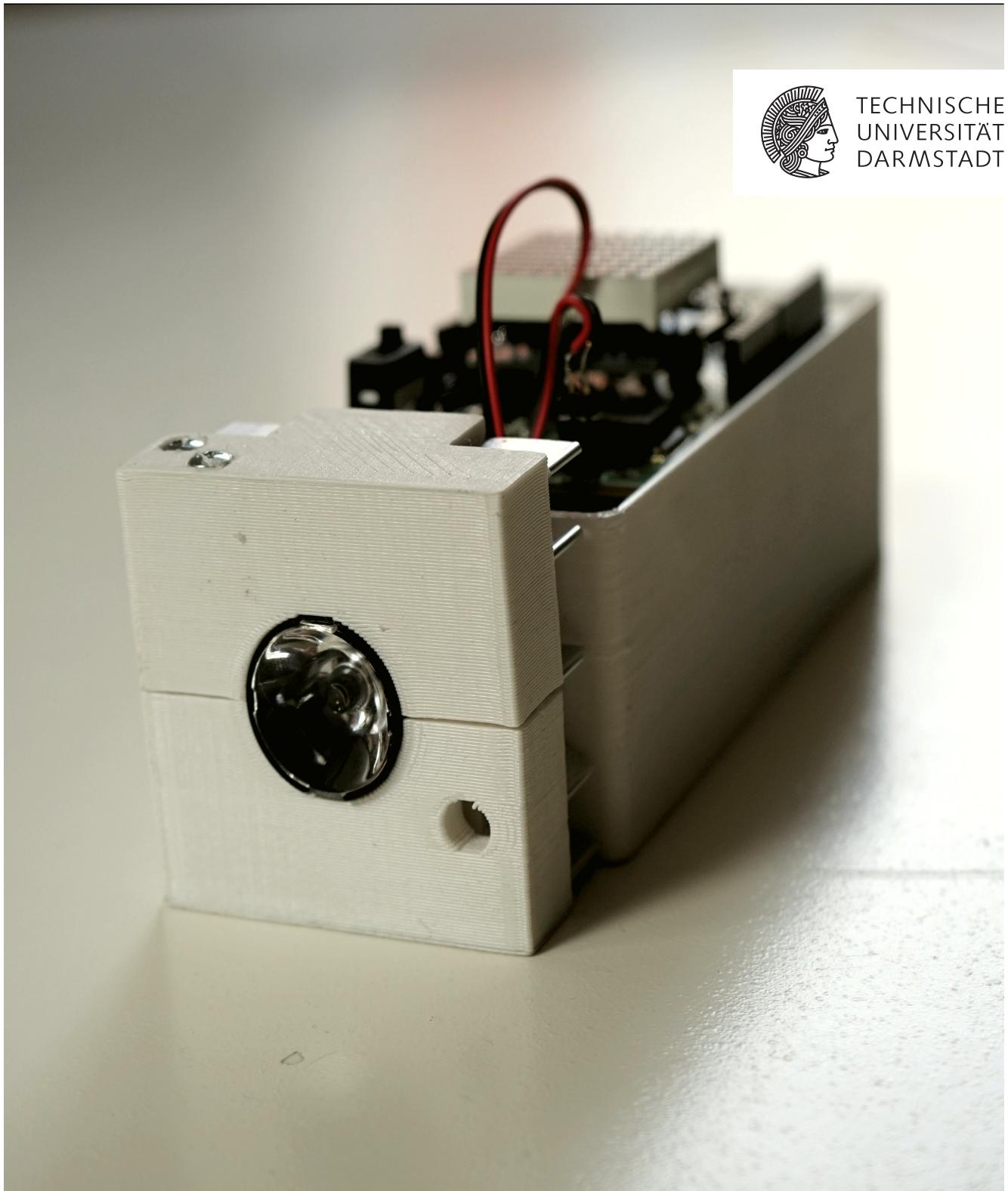


# VLC Workshop

Workshop zur Kommunikation mit sichtbarem Licht am Fachgebiet Lichttechnik



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## 1 Einführung

Für diesen Workshop werden wir einen Arduino Uno mit unserem VLC-Shield programmieren, um mit sichtbarem Licht (Visible Light) zwischen zwei Einheiten zu kommunizieren (Communication). Dazu werden wir die Arudino IDE nutzen. Sie ermöglicht eine einfache Programmierung von unterstützten Microcontrollern.

### 1.1 Die Arduino Programmierumgebung

Um mit der Programmierung unserer Microcontroller anfangen zu können müssen wir zuerst die Entwicklungs-umgebung starten. Diese befindet sich direkt auf dem Desktop der Laptops oder kann Zuhause unter <https://www.arduino.cc/en/Main/Software> kostenfrei runtergeladen werden. Wenn ihr die Umgebung gestartet habt müsst ihr noch sicherstellen, dass das richtige Board ausgewählt ist. Dieses findet ihr in **Werkzeuge > Board > Arduino/Genuino Uno**.

### 1.2 Programmierung

Grundsätzlich besteht ein Programm für Arduino aus mindestens zwei Funktionen:

- **void setup()** {} - Diese Funktion wird als Initailisierung **einmal** zu Beginn des Programms durchlaufen. Hier können bspw. die Ein- und Ausgänge des Controllers definiert werden.
- **void loop()** {} - Diese Funktion wird zur Laufzeit des Programmes **ständig** in einer Endlosschleife aufgerufen. Hier sollte die eigentliche Logik der Anwendung implementiert werden.

Im Anhang dieser Anleitung könnt ihr eine kleine Referenz zur Programmierung und zu den für diesen Workshop wichtigsten Funktionen finden. Über die Arduino IDE könnet ihr auch **Beispielprogramme öffnen** und verwenden. Ein solches Beispiel, welches eine LED zum blinken bringt findet ihr unter **Datei > Beispiele > 01.Basics > Blink**.

```
1 // die setup Funktion wird am Anfang einmal ausgeführt
2 void setup() {
3     // die blaue LED auf Pin 5 wird als Ausgang konfiguriert
4     pinMode(LED_BUILTIN, OUTPUT);
5 }
6
7 // die loop Funktion wird wieder und wieder für immer ausgeführt
8 void loop() {
9     digitalWrite(LED_BUILTIN, HIGH);      // schalte die LED ein (setze den Pegel von Pin 5 auf HIGH)
10    delay(1000);                      // warte 1000ms (eine Sekunde)
11    digitalWrite(LED_BUILTIN, LOW);     // schalte die LED aus (setze den Pegel von Pin 5 auf LOW)
12    delay(1000);                      // warte wieder 1000ms
13 }
```

### 1.3 Hochladen

Um nun die LED des Arduino blinken zu lassen müsst ihr das geschriebene auf das Board laden. Hierzu müsst ihr es an den Laptop anschließen und gegebenenfalls noch unter **Werkzeuge > Port** auswählen. Anschließend könnet ihr es über die runde Schaltfläche oben links, in der ein Pfeil nach rechts zu sehen ist hochladen. Alternativ könnet ihr das auch in **Sketch > Hochladen** machen. Sollten beim Kompilieren oder Übertragen irgendwelche Fehler aufgetreten sein könnet ihr sie im unteren Ausgabefenster sehen.

Nun sollte die LED auf dem Board blinken.

## 2 Komponenten des VLC Microcontrollers

Für diesen Workshop haben wir den Arduino etwas erweitert, um mit ihm die Kommunikation mit sichtbarem Licht zu ermöglichen. In Abbildung 1 könnt ihr die Bestandteile sehen:

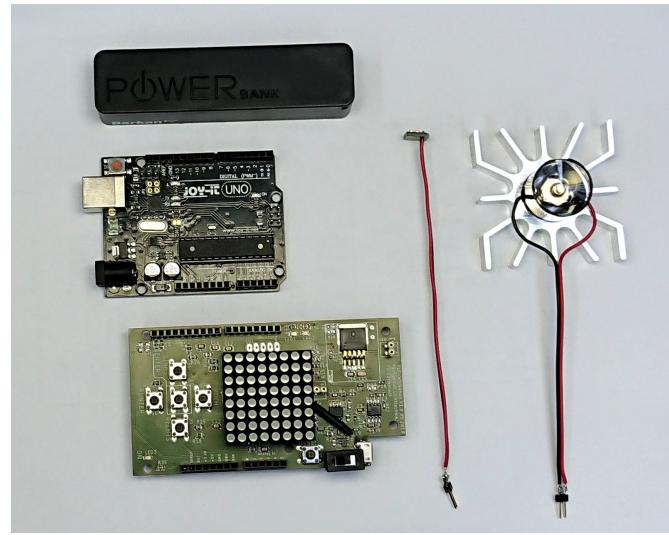


Abbildung 1: Die Komponenten des VLC Microcontrollers (ohne Gehäuse)

Zu sehen sind:

- Oben links die Powerbank
- Darunter der Arduino Uno
- Unten links das VLC Shield
- Ganz rechts die LED mit Kühlkörper
- Links davon die Fotodiode

Auf dem Shield selbst befinden sich fünf Taster (Tasten), zwei LEDs, ein LED Display, ein Reset Taster und ein An- und Ausschalter. Das Shield kann einfach auf den Arduino gesteckt werden.

## 3 Zusammenbau des VLC Microcontrollers

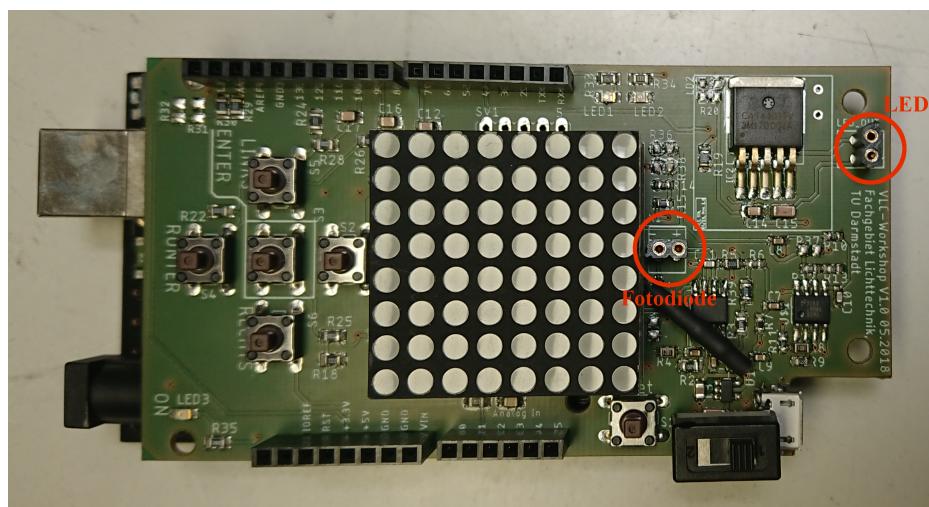


Abbildung 2: Das VLC Shield mit den Anschlüssen für die Fotodiode und die LED markiert.

Den Controller baut ihr wie folgt zusammen:

Die Powerbank kommt unten in das Gehäuse. Darauf kommt der Arduino, den ihr mit dem Kabel an die Powerbank anschließen könnt. Darauf wird das VLC Shield gesteckt. Danach könnt ihr die LED und Fotodiode vorne im Gehäuse an die vorgesehenen Stellen platzieren. Anschließend werden die Fotodiode und die LED an das Shield angeschlossen, wie in Abbildung 2 zu sehen. Achtet dabei bitte darauf, dass das rote Kabel an + und das Schwarze an - angeschlossen wird. **WICHTIG!:** Die Anschlüsse für die Fotodiode und die LED dürfen auf keinen Fall vertauscht werden!

## 4 Einstiegsaufgabe - Blink 2

### 4.1 Blink mit dem VLC Shield

Nun stecken wir das VLC Shield auf den Microcontroller. Dadurch wird die auf dem Arduino eingebaute LED verdeckt. Wollen wir nun eine andere blinkende LED sehen, müssen wir hierzu die entsprechende LED ansteuern. Das Shield hat hiervon gleich zwei: eine Blaue auf Pin 5 und eine Rote auf Pin 6. Wir wollen also jetzt das Blink Programm für die blaue LED umschreiben. Hierzu muss nur überall im bestehenden Programm `LED_BUILTIN` durch 5 ersetzt werden. Dadurch wird nun nicht mehr die definierte LED, sondern die blaue LED, sie an Port 5 angeschlossen ist, angesteuert.

### 4.2 Blink mit zwei LEDs

Wir wollen jetzt auch noch die rote LED auf dem Board abwechselnd mit der Blauen blinken lassen. Konfiguriert sie hierzu als Ausgang und schaltet sie immer dann an, wenn ihr die blaue LED ausschaltet und umgekehrt.

## 5 Taster

Wir wollen nun ein weiteres externes Element in unsere Steuerung aufnehmen: die Taster. Auf unserem Board befinden sich unter dem Display fünf davon. Wir wollen nun den Mittleren dazu nutzen die blaue LED einzuschalten, wenn er gedrückt wird.

### 5.1 Eingänge

Um einen Pin als Eingang zu verwenden, muss er zunächst als Eingang gesetzt werden. Dazu wird `pinMode` verwendet. Um später den aktuellen Zustand zu lesen, wird `digitalRead` verwendet. Wenn der Pin gerade mit "Ground"(der Erdung) verbunden ist liefert diese Funktion 0, bzw. logisch falsch zurück. Ist er gerade mit der Spannungsversorgung (bzw. High) verbunden, liefert sie 1, bzw. logisch wahr zurück. Ist der Pin überhaupt nicht verbunden, dann liefert `digitalRead` ziemlich zufällig 0 oder 1. Da sich mit zufälligen Werten schlecht rechnen lässt, sollte man darauf achten, dass das nie passiert. Das kann allerdings auf unserem Board nicht geschehen, da alle Taster bereits verbunden sind. Der "ENTER" Taster in der Mitte, den wir nutzen möchten ist mit Pin 8 verbunden.

### 5.2 Die Implementierung

Hier könnt ihr nun ein kleines Beispielprogramm zum Taster finden. Darin wird zuerst mit einer Konstanten festgelegt, auf welchem Pin sich der Taster befindet. Dann wird dieser in der `setup` Funktion als Input (also Eingang) initialisiert. Anschließend haben wir eine kleine `if`-Bedingung für euch vorbereitet. In ihr wird eine Bedingung abgefragt. Sollte diese wahr sein, wird der direkt folgende Teil in den geschweiften Klammern ausgeführt. Optional kann dann noch ein `else` Pfad hinzugefügt werden. Dieser Teil wird ausgeführt, sollte die Bedingung falsch gewesen sein.

```
1 /**
2 * Programm Taster
3 */
4
5 const int tasterPin = 8;
6
7 void setup() {
8     // hier weitere Pins initialisieren...
9
10    pinMode(tasterPin, INPUT);
11}
12
13 /**
14 * Diese Funktion wird in einer Schleife durchlaufen
15 */
16 void loop() {
17    // hier kann mit digitalRead(8) der Pegel von Pin 8 abgefragt werden
```

```

18 // der Pegel kann HIGH oder LOW sein
19 if(...) {
20     // TODO
21 } else {
22     // TODO
23 }
24 }
```

## 6 Serielle Kommunikation

In diesem Workshop werden wir oft serielle Kommunikation mit dem Display benötigen. Zu diesem Zweck gibt es nun eine kleine Einführung. Ihr könnt sie auch dazu nutzen, um euch Nachrichten vom Arduino auf dem Laptop anzeigen zu lassen. Zuerst müssen wir in der `setup` Funktion die serielle Kommunikation initialisieren. Dies machen wir mit Hilfe des Befehls `Serial.begin(9600)`. Er startet auf dem Arduino eine serielle Kommunikation mit einer sogenannten Baud-Rate von 9600. Das bedeutet, dass er über seine serielle Schnittstelle 9600 Zeichen pro Sekunde senden wird. Mit `Serial.print` und `Serial.println` könnt ihr nun Strings (also Text) senden. Der Text muss in Anführungszeichen an die Funktion übergeben werden. Um diesen Text dann lesen zu können müsst ihr den seriellen Monitor öffnen. Diesen könnt ihr entweder oben rechts in der Programmierumgebung oder unter `Werkzeuge > Serieller Monitor` finden.

Versucht nun in einem kleinen Programm eine kurze Nachricht einmalig in der `setup` Funktion zu senden.

Danach könnt ihr versuchen, eine weitere Nachricht in der `loop` Funktion zu senden um zu sehen was passiert.

## 7 Github Bibliotheken herunterladen

Eine der Bibliotheken, die ihr benötigt um den Kontroller zu programmieren haben wir auf Github hochgeladen. Dort ist sie zu finden unter <https://github.com/Lithimlin/VLC-Transciever>. Der Link steht auch nochmal am Ende dieses Dokuments. Dort sind auch noch weitere Bibliotheken zu finden, die ihr benötigen werdet.

Auf dieser Webseite könnt ihr rechts über der Dateiliste einen grünen Button `Clone or Download` finden. Ihr könnt einen .zip-Ordner der Bibliothek herunterladen, indem ihr `Clone or Download > Download ZIP` auswählt. Im folgenden wird beschrieben, wie ihr .zip-Ordner als Bibliotheken einbinden könnt.

## 8 Bibliotheken

Bibliotheken enthalten vorprogrammierte Funktionen, die genutzt werden können und somit von euch nicht implementiert werden müssen.

Sie können ganz leicht mit dem Befehl `#include <...>` in ein Programm eingebunden werden. Damit das funktioniert müssen sie allerdings erst im *libraries* Ordner der Arduino IDE sein. Um eine Bibliothek zu importieren könnt ihr ganz einfach in der Programmierumgebung auf `Sketch > Bibliothek einbinden > .ZIP-Bibliothek einbinden` einen .zip-Ordner einbinden. Am Ende dieses Dokuments befinden sich die Links zu den Bibliotheken die ihr brauchen werdet. Für die nächste Aufgabe müsst ihr die `Adafruit_GFX` und die `Max72xxPanel` Bibliotheken einbinden.

## 9 Ein erster Punkt

Wie bereits erwähnt, kommuniziert der Microcontroller über eine serielle Schnittstelle mit dem Display. Damit das funktioniert müsst ihr neben den Bibliotheken `Adafruit_GFX` und `Max72xxPanel` noch die Bibliothek `SPI` einbinden.

Danach können wir anfangen, auf das Display zu zeichnen. Hierzu wird im folgenden Code ein Display initialisiert und dann mit der Intensität und Rotation konfiguriert. Versucht an der angegebenen Stelle im Code ein beliebiges Pixel des Displays einzuschalten. Das Display wurde hier `matrix` genannt. Hierzu könnt ihr im Anhang eine Referenz für das Display finden.

```

1 #include <SPI.h>
2 #include <Adafruit_GFX.h>
3 #include <Max72xxPanel.h>
4
5 // Das Display auf dem Matrix Pin 2 mit einer Höhe und Breite von je einem Display initialisieren
6 Max72xxPanel matrix = Max72xxPanel(2, 1, 1);
7
8 void setup() {
9     // Serielle Kommunikation mit dem Display starten
10    Serial.begin(9600);
11    // Das Display konfigurieren
12    matrix.setIntensity(5);
13    matrix.setRotation(0, 3);
14    // Das Display leeren
```

```

15 matrix.fillRect(LOW);
16 matrix.write();
17
18 //Euer Code hier//
19 }
20
21 void loop() {
22 }

```

## 10 Ein Zeichen in Punkten

Nun wollen wir versuchen, ein Zeichen mittels der Funktion `drawChar(...)` auf das Display zu zeichnen. Hierzu könnt ihr den gleichen Code wie oben verwenden. Mit einem Zeichen ist hier ein Buchstabe oder Satzzeichen gemeint.

Wenn ihr es geschafft habt, ein Zeichen zu schreiben, könnt ihr versuchen das Ganze auf Knopfdruck geschehen zu lassen.

## 11 Noch mehr Bibliotheken

Für den letzten Abschnitt benötigen wir noch zwei weitere Bibliotheken: `VLC-Transceiver` und `Timer`. Auf dem Desktop könnt ihr auch diese zwei Bibliotheken finden. Bindet sie genau wie die beiden Display Bibliotheken ein. Bitte beachtet hierbei, dass ihr die `VLC-Transceiver` Bibliothek über `Sketch > Bibliotheken einbinden > VLC-Transceiver` einbindet, da es hierbei nicht nur eine Datei ist.

## 12 Die LED in Betrieb nehmen

Wir wollen nun kurz versuchen die große LED vorne im Gehäuse einzuschalten bevor wir damit Daten senden. Die "Transmitter"-LED liegt auf Pin 4, ist aber auch durch die Einbindung der `Constants.h` als Konstante hinterlegt, ähnlich wie `HIGH` und `LOW`. Der Name dieser Konstante ist `TRANSMITTER_PIN`. Die Namen aller Konstanten können auch hinten in der Referenz zur `VLC-Transceiver` Bibliothek nochmal eingesehen werden. Dort findet ihr auch für später die wichtigsten Funktionen aufgelistet und erklärt. Bringt also nun die "Transmitter"-LED zum Blinken.

## 13 Timer

Jetzt würden wir gerne die LED alle 500ms an und dann wieder ausschalten. Um uns gleichzeitig aber noch auf der Matrix Dinge anzeigen lassen zu können müssen wir es vermeiden mit der `delay()` Funktion zu arbeiten, da in dieser Zeit keine Befehle ausgeführt werden können. Hier würden wir aber gerne gleichzeitig einen Knopf nutzen, um mit ihm ein Bild auf dem Display anzeigen zu lassen.

```

1 #include <Timer.h>
2
3 #include <SPI.h>
4 #include <Adafruit_GFX.h>
5 #include <Max72xxPanel.h>
6
7 #include <Constants.h>
8 #include <Transceiver.h>
9 #include <LEDBitmap.h>
10
11 // Das Display auf dem Matrix Pin mit einer Höhe und Breite von je einem Display initialisieren
12 Max72xxPanel matrix = Max72xxPanel(MATRIX_PIN, 1, 1);
13
14 // Ein Datenarray initialisieren
15 int data[] = {1, 1, 1, 0, 0, 1, 1, 1,
16   1, 0, 0, 0, 1, 0, 0, 0,
17   1, 1, 0, 0, 1, 0, 0, 1,
18   1, 0, 0, 0, 0, 1, 1, 1,
19   0, 1, 0, 0, 0, 0, 0, 0,
20   0, 1, 0, 0, 1, 1, 1, 0,
21   0, 1, 0, 0, 0, 1, 0, 0,
22   0, 1, 1, 1, 0, 1, 0, 0};
23 // Hier wurde ein sogenanntes "Array" initialisiert. Es ist eine Art Liste, die in diesem Fall Integers
24 // enthält
25 // Mit dem Datenarray ein Bild initialisieren
26 LEDBitmap bild(8, 8, data);
27 // Einen Timer initialisieren

```

```

27 Timer t
28
29 void setup() {
30     // Serielle Kommunikation mit dem Display starten
31     Serial.begin(9600);
32     // Das Display konfigurieren
33     matrix.setIntensity(5);
34     matrix.setRotation(0, 3);
35     // Das Display leeren
36     matrix.fillRect(0, 0, 8, 8, LOW);
37     matrix.write();
38     // Ein Ereignis erstellen, das alle 500ms die LED ein und ausschaltet
39     int8_t blinkEvent = t.every(/*Zeit in ms*/, /*Funktion, die ausgeführt werden soll*/, (void*)0);
40 }
41
42 void loop() {
43     if( /*Bedingung hier einfügen*/ ) {
44         matrix.drawBitmap(0, 0, bild.getBitmap(), 8, 8, HIGH); // Das Bild als Bitmap auf das Display malen
45         matrix.write(); // Das Display aktualisieren
46     } else {
47         matrix.fillRect(0, 0, 8, 8, LOW); // Das Display leeren
48         matrix.write(); // Das Display aktualisieren
49     }
50     t.update(); // den Timer aktualisieren
51 }
52
53 void /*Name der Funktion, die vom Timer ausgeführt werden soll*/() {
54     if(digitalRead(TRANSMITTER_PIN) == HIGH) { // Wenn die LED eingeschaltet ist...
55         digitalWrite(TRANSMITTER_PIN, LOW); //... schalte sie aus
56     } else { // Ansonsten...
57         // Hier den Code zum LED einschalten einfügen
58     }
59 }
```

An der Stelle

```

1 void /*Name der Funktion, die vom Timer ausgeführt werden soll*/() {
2     // Code hier einfügen
3 }
```

wird hierbei eine neue Funktion erstellt, damit sie an den Timer übergeben werden kann. Funktionen werden immer wie folgt definiert:

TYP name(Parameter){code der ausgeführt werden soll}

Parameter sind hierbei optional und im oberen Beispiel nicht gebraucht. Sollte man allerdings Parameter an eine Funktion übergeben wollen müssen diese wie eine Variable mit TYP name definiert werden. Hier ein kleines Beispiel:

```

1 int addieren(int zahl1, int zahl2) {
2     return zahl1 + zahl2;
3 }
```

Oben wird eine Funktion "addieren" definiert. Sie hat den Typ **int**, also Integer, und nimmt zwei **int** Parameter: **zahl1** und **zahl2**. Mit **return** wird das Nachfolgende zurückgegeben. In diesem Fall ist das **zahl1 + zahl2**.

Im Code oben wurde bereits eine Funktion vom Typ **void** angefangen zu definieren. **void** heißt, dass die Funktion keine Rückgabe, also kein **return** benötigt.

## 14 Bibliotheksbeispiele

Ähnlich wie beim Beispiel **Blink** gibt es auch für Bibliotheken Beispielprogramme. Ladet nun auf einen Controller das Beispiel **Transmitter** und auf einen anderen das Beispiel **Reciever**. Diese könnt ihr in **Datei > Beispiele > VLC-Transciever** finden. Damit könnt ihr dann Daten senden. Bitte beachtet, dass für diesen Workshop Lücken in den Beispielen sind, die ihr füllen müsst.

```

1 // --- Transmitter (Sender) --- //
2 #include <Timer.h> // Die Timer Bibliothek mit einbinden
3
4 #include <Constants.h>
5 #include <LEDBitmap.h>
6 #include <Transciever.h>
```

```

7 // Den Transciever initialisieren
8 Transciever transciever;
9
10 // Ein Datenarray initialisieren
11 int data[] = {1, 1, 1, 0, 0, 1, 1, 1,
12 1, 0, 0, 1, 0, 0, 0,
13 1, 1, 0, 0, 1, 0, 0, 1,
14 1, 0, 0, 0, 1, 1, 1,
15 0, 1, 0, 0, 0, 0, 0, 0,
16 0, 1, 0, 0, 1, 1, 1, 0,
17 0, 1, 0, 0, 0, 1, 0, 0,
18 0, 1, 0, 0, 1, 0, 0, 0,
19 0, 1, 1, 1, 0, 1, 0, 0};
20 // Mit dem Datenarray ein Bild initialisieren
21 LEDBitmap image(8, 8, data);
22 // Einen Timer initialisieren
23 Timer t;
24
25 void setup() {
26     // Die serielle Kommunikation mit dem Display starten
27     // !!! Code hier einfügen !!! //
28     // Ein Event an den Timer binden, das alle 5 Sekunden "sendData" ausführt
29     int8_t sendEvent = t.every(5 * 1000, sendData, (void*)0);
30     // Den Transmitter starten
31     // !!! Code hier einfügen !!! //
32 }
33
34 void loop() {
35     // Den Timer aktualisieren
36     // !!! Code hier einfügen !!! //
37 }
38
39 void sendData(void){
40     // Das vorher initialisierte Bild senden
41     // !!! Code hier einfügen !!! //
42 }

```

```

1 // --- Reciever (Empfänger) --- //
2 #include <SPI.h>
3 #include <Adafruit_GFX.h>
4 #include <Max72xxPanel.h>
5
6 #include <Constants.h>
7 #include <Transciever.h>
8 #include <LEDBitmap.h>
9
10 // Das Display auf dem Matrix Pin mit einer Höhe und Breite von je einem Display initialisieren
11 Max72xxPanel matrix = Max72xxPanel(MATRIX_PIN, 1, 1);
12 // Den Transmitter initialisieren
13 Transciever transciever;
14 // Eine Variable für die letzte Empfangszeit initialisieren
15 long lastReception;
16
17 void setup() {
18     // Serielle Kommunikation mit dem Display starten
19     Serial.begin(9600);
20     // Das Display konfigurieren
21     matrix.setIntensity(5);
22     matrix.setRotation(0, 3);
23     // Das Display leeren
24     // !!! Code hier einfügen !!! //
25     // Den Reciever starten
26     // !!! Code hier einfügen !!! //
27
28     lastReception = millis(); //Den Startwert für die letzte Empfangszeit auf den Einschaltzeitpunkt legen
29 }
30
31 void loop() {
32     // Wenn ein Bild erfolgreich empfangen wurde...
33     if(transciever.receptionSuccessful()) {

```

```
34     lastReception = millis(); //... speichere die Zeit, ...
35     matrix.fillRect(0, 0, 128, 64, LOW); //... leere das Display...
36     matrix.drawBitmap(0, 0, /* !!! Code hier einfügen !!! */, 8, 8, HIGH, LOW); //... und zeige das
37     // empfangene Bild an.
38     matrix.write();
39 }
40 if((millis() - lastReception) >= 5000) { // Wenn seit 5 Sekunden oder mehr kein Bild mehr empfangen
41     // wurde...
42     matrix.fillRect(0, 0, 128, 64, LOW); //... leere das Display
43     matrix.write();
44 }
```

**Hinweis:** ein empfangenes Bild muss noch immer erst in eine Bitmap umgewandelt werden, bevor es auf das Display gemalt werden kann.

## 15 Referenz

### 15.1 C-Syntax

- Variablen Deklaration

```
TYP name = Wert;
```

Hier wird eine Variable vom Typ TYP, mit Namen name und dem Wert Wert deklariert. Der Typ muss nur bei der ersten Zuweisung dabei stehen. Bei jeder weiteren Zuweisung muss der Typ weggelassen werden. Also zum Beispiel

```
= Wert2;
```

Weiteres Beispiel:

```
1 int counter = 0;  
2 ...  
3 counter = 3;
```

- IF-Bedingung

```
if( Bedingung ) { } else { }
```

Hier findet eine Verzweigung des sogenannten Kontrollflusses statt. Das heißt, dass nur einer der angegebenen Pfade ausgeführt wird. Wenn die Bedingung zutrifft, wird der Pfad im ersten Set gesweifter Klammern ausgeführt. Ist sie jedoch nicht erfüllt, so wird der Pfad im zweiten Set geschweifter Klammern nach dem else ausgeführt. Der zweite Pfad ist hierbei optional.

Beispiel:

```
1 boolean kontrolle = false;  
2 ...  
3 if(kontrolle == true) {  
4     digitalWrite(LED_BUILTIN, HIGH); //Dieser Pfad wird nicht ausgeführt, da kontrolle den Wert  
5         false hat, und somit nicht true ist.  
6 } else {  
7     digitalWrite(LED_BUILTIN, LOW); //Dieser Pfad ist der, der ausgeführt wird  
8 }  
9 ...  
10 boolean kontrolle2 = false;  
11 ...  
12 if(kontrolle2 == true) {  
13     digitalWrite(LED_BUILTIN, LOW); //Hier wird dieser Pfad nicht ausgeführt, da kontrolle2 den  
14         Wert false, also nicht den Wert true hat.  
15 } //Da allerdings keine "else"-Pfad existiert, wird er einfach übersprungen.
```

- While-Schleife

```
while(Bedingung){ }
```

While-Schleifen wiederholen den in ihnen enthaltenen Code solange die Bedingung erfüllt ist. Beispiel:

```
1 int counter = 3;  
2 ...  
3 while(counter > 0) { //Hier wird geprüft, ob die Variable counter größer als 0 ist.  
4     counter = counter - 1; //Dadurch wird von der Variable counter solange 1 abgezogen, bis sie  
5         kleiner/gleich 0 ist.  
6 }
```

- For-Schleife

```
for(Definition; Bedingung; Anweisung){ }
```

For-Schleifen definieren eine Variable, die sich bei jeder Wiederholung entsprechend der Anweisung solange die Bedingung erfüllt ist. Beispiel:

```
1 for(int counter = 0; counter < 10; counter++) { //Diese Schleife wird so oft ausgeführt bis  
2     counter größer/gleich 10 ist. Bei jeder Wiederholung counter um 1 erhöht durch "counter++".  
3     Serial.println("Hallo"); //In jeder Wiederholung wird dieser Code ausgeführt  
4 }
```

---

## 15.2 Datentypen

---

- **int**  
Integers (oder ints) bezeichnen ganze Zahlen.
- **double**  
Doubles halten Kommazahlen. Sie werden hier allerdings nicht benötigt werden.
- **long**  
Longs beschreiben sehr große Zahlen. Die Arduino Befehle `millis()` oder `micros()` geben beispielsweise einen Long zurück.
- **bool**  
Booleans stellen Wahrheitswerte dar und können entweder `true`, also wahr oder `false`, also falsch sein.
- **String**  
Strings repräsentieren Text. Dieser Text muss in Anführungszeichen stehen.
- **const**  
Consts bezeichnen Konstanten, die während des Programmverlaufs nicht geändert werden können. Sie benötigen noch einen weiteren Datentyp um zu spezifizieren, welche Daten wie speichern sollen.

---

## 15.3 Arduino Befehle

---

- **void setup()**  
Diese Funktion wird als Initialisierung einmal zu Beginn des Programms durchlaufen. Hier können die Ein- und Ausgänge des Controllers definiert werden
- **void loop()**  
Diese Funktion wird zur Laufzeit des Programmes ständig in einer Endlosschleife aufgerufen. Hier sollte die eigentliche Logik der Anwendung implementiert werden.
- **void pinMode(pin, mode)**  
Konfiguriert das Verhalten eines Pins, der Modus kann entweder INPUT oder OUTPUT sein.
- **void digitalWrite(pin, value)**  
Legt den Wert HIGH oder LOW auf einen Pin.
- **int digitalRead(pin)**  
Liest den Zustand eines Pins aus und gibt diesen zurück.
- **void analogWrite(pin, value)**  
Schreibt einen analogen Wert zwischen 0 und 255 als PWM-Signal auf einen Ausgang.
- **void delay(millis)**  
Wartet den übergebenen Wert in Millisekunden.
- **void delayMicroseconds(micros)**  
Wartet den übergebenen Wert in Mikrosekunden.

## 16 Die Max72xxPanel Bibliothek für das Display

Hier sind einmal die wichtigsten Funktionen für das Display aufgeführt.

- `Max72xxPanel name = Max72xxPanel(int pin, int anzahlHorizontal, int anzahlVertikal)`  
Der sogenannte Konstruktor eines “Max72xxPanel” Displays. Mit ihm initialisiert man ein neues Display auf dem übergebenen Pin. Die beiden anderen Werte sind die Anzahl der Displays, die horizontal und vertikal miteinander verbunden sind. In unserem Fall sind beide dieser Werte 1
- `setRotation(int displayNummer, int anzahlRotationen)`  
Rotiert das Display mit der gegebenen Nummer (es wird bei 0 angefangen zu zählen) so oft um 90° , wie im zweiten Parameter angegeben.
- `setIntensity()`  
Setzt die Intensität der Displays. Der Wert kann zwischen 0 und 15 liegen. Wir empfehlen Werte zwischen 3 und 5
- `drawPixel(int x, int y, int Farbe)`  
Setzt das Pixel an der angegebenen Stelle auf die angegebene Farbe. Für dieses Display kann die Farbe entweder HIGH oder LOW (also an oder aus) sein.
- `drawLine(int x0, int y0, int x1, int y1, int Farbe)`  
Malt eine Linie vom Punkt (x0, y0) zum Punkt (x1, y1) in der gegebenen Farbe.
- `drawBitmap(int x, int y, int[] Bitmap, int Breite, int Höhe, int Farbe)`  
Malt die angegebene Bitmap von der Stelle (x, y) aus in der angegebenen Farbe auf das Display. Breite und Höhe bezeichnen die Dimensionen der Bitmap.
- `drawChar(int x, int y, char Zeichen, int Farbe, int Hintergrundfarbe, int Größe)`  
Malt ein Zeichen (einen Buchstaben) von der angebenden Stelle (x, y) in der angegebenen Farbe mit der angegebenen Hintergrundfarbe auf das Display. Der letzte Parameter ist eine Skalierung des Zeichens. Auf unser Display passen nur Zeichen der Größe 1
- `fillScreen(int Farbe)`  
Füllt das gesamte Display mit der angegebenen Farbe
- `write()`  
Erst diese Funktion schreibt die gemachten Änderungen auf das Display, sodass diese angezeigt werden

---

## 17 Die VLC-Transciever Bibliothek

---

### 17.1 Transciever

---

- **startTransmitter()**  
Startet den Transmitter
- **stopTransmitter()**  
Stoppt den Transmitter
- **startReceiver()**  
Startet den Reciever
- **stopReceiver()**  
Stoppt den Reciever
- **sendData(LEDBitmap Bild)**  
Sendet über den Transmitter eine 8x8 LEDBitmap
- **transmitterIsStarted()**  
Gibt einen Boolean zurück. Er ist wahr, wenn der Transmitter gestartet wurde und falsch sonst
- **lastTransmissionComplete()**  
Gibt einen Boolean zurück. Er ist wahr, wenn zur Zeit keine Übertragung läuft
- **recieverIsStarted()**  
Gibt einen Boolean zurück. Er ist wahr, wenn der Reciever gestartet wurde
- **isRecieving()**  
Gibt einen Boolean zurück. Er ist wahr, wenn gerade eine Übertragung empfangen wird
- **receptionSuccessful()**  
Gibt einen Boolean zurück. Er ist wahr, wenn eine Übertragung erfolgreich empfangen wurde
- **hadReceptionError()**  
Gibt einen Boolean zurück. Er ist wahr, wenn es bei dem Empfangen einer Übertragung einen Fehler gab.
- **getImage()**  
Gibt die zuletzt empfangene 8x8 LEDBitmap zurück.

---

### 17.2 LEDBitmap

---

- **LEDBitmap name(int Breite, int Höhe, int[] Bild);**  
Der sogenannte Konstruktor einer LEDBitmap. Mit ihm erstellt man eine neue LEDBitmap. Die zu übergebenden Parameter sind die Breite und die Höhe des Bildes sowie ein Integer Array, in dem das Bild hinterlegt ist. Ein Beispiel hierfür kann im Bibliotheksbeispiel `Transmitter` gefunden werden.
- **getBitmap()**  
Gibt ein Array für die Display-Funktion `drawBitmap(int[] Bitmap)` zurück
- **getPixelValue(int x, int y)**  
Gibt einen Integer zurück, der den Wert der LEDBitmap an der Stelle (x (Spalte), y (Zeile)) von oben links aus gesehen repräsentiert
- **setPixelValue(int x, int y, int Wert)**  
Setzt den Wert einer LEDBitmap an der Stelle (x, y) auf den angegebenen Wert (0 oder 1)
- **getHeight()**  
Gibt die Höhe einer LEDBitmap als Integer zurück
- **getWidth()**  
Gibt die Breite einer LEDBitmap als Integer zurück

- `getSize()`  
Gibt die Gesamtgröße einer LEDBitmap als Integer zurück
- `clear()`  
Setzt alle Einträge einer LEDBitmap auf 0

---

### 17.3 Constants

---

- `BUTTON_UP`  
Der Pin des Tasters “HOCH”
- `BUTTON_DOWN`  
Der Pin des Tasters “RUNTER”
- `BUTTON_LEFT`  
Der Pin des Tasters “LINKS”
- `BUTTON_RIGHT`  
Der Pin des Tasters “RECHTS”
- `BUTTON_ENTER`  
Der Pin des Tasters “ENTER”
- `LED_STATUS_1` oder `BLUE`  
Der Pin der blauen LED
- `LED_STATUS_2` oder `RED`  
Der Pin der roten LED
- `MATRIX_PIN`  
Der Pin des LED Displays

---

### 18 Kontakt

---

Jonas Kobbert: [kobbert@lichttechnik.tu-darmstadt.de](mailto:kobbert@lichttechnik.tu-darmstadt.de)

Sebastian Beck: [beck@lichttechnik.tu-darmstadt.de](mailto:beck@lichttechnik.tu-darmstadt.de)

Julian Euler: [julian.euler@stud.tu-darmstadt.de](mailto:julian.euler@stud.tu-darmstadt.de)

Yvonne Späck-Leigsnering: [sphaeck@temf.tu-darmstadt.de](mailto:sphaeck@temf.tu-darmstadt.de)

---

### 19 Links

---

VLC Bibliothek: <https://github.com/Lithimlin/VLC-Transciever>

Adafruit\_GFX Bibliothek: <https://github.com/adafruit/Adafruit-GFX-Library>

Max72xxPanel Bibliothek: <https://github.com/markruys/arduino-Max72xxPanel>

Timer Bibliothek: <https://github.com/JChristensen/Timer>