

# Dokumentation der Arduino Photokommunikation

von Tom Arlt, Florian Körwer, Florian Meyer, Henning Müller,  
Cong Tai Phan

Hochschule Niederrhein FB03, der 03.01.2021

---

## Links

- [GitHub link](#)
- [IDE3 link](#)

## Einleitung

Unsere Aufgabe war es als Gruppe für das Erstsemesterprojekt, im folgenden "ESP" genannt, zwei Arduinos mithilfe von Leds und Photoresistoren miteinander kommunizieren zu lassen und unsere Arbeit hiermit zu dokumentieren.

Die notwendigen Bauteile wurden uns hierfür pro Person von der Hochschule zur Verfügung gestellt:

- 1x Arduino Uno
- 2x LM392 mit Photoresistor und Potenziometer
- 1x Breadboard
- 2x 330 Ohm Widerstände
- verschied. LEDs
- verschied. Kabel für das Breadboard
- ~~1x Potenziometer~~
- ~~1x Ultraschallsensor~~

Da der Wissensstand innerhalb der Gruppe zum Thema Arduino sehr verschieden war, wurde nur zeitgleich und immer gemeinsam gearbeitet. Die Teilnehmenden, welche bereits einiges an Erfahrung mit sich brachten, fungierten eher als Lehrpersonen, die den anderen das nötige Wissen überschaubar vermittelt haben, sodass alle dann gemeinsam die Aufgaben bearbeiten konnten.

## Lösung der Aufgabe

### Strategie

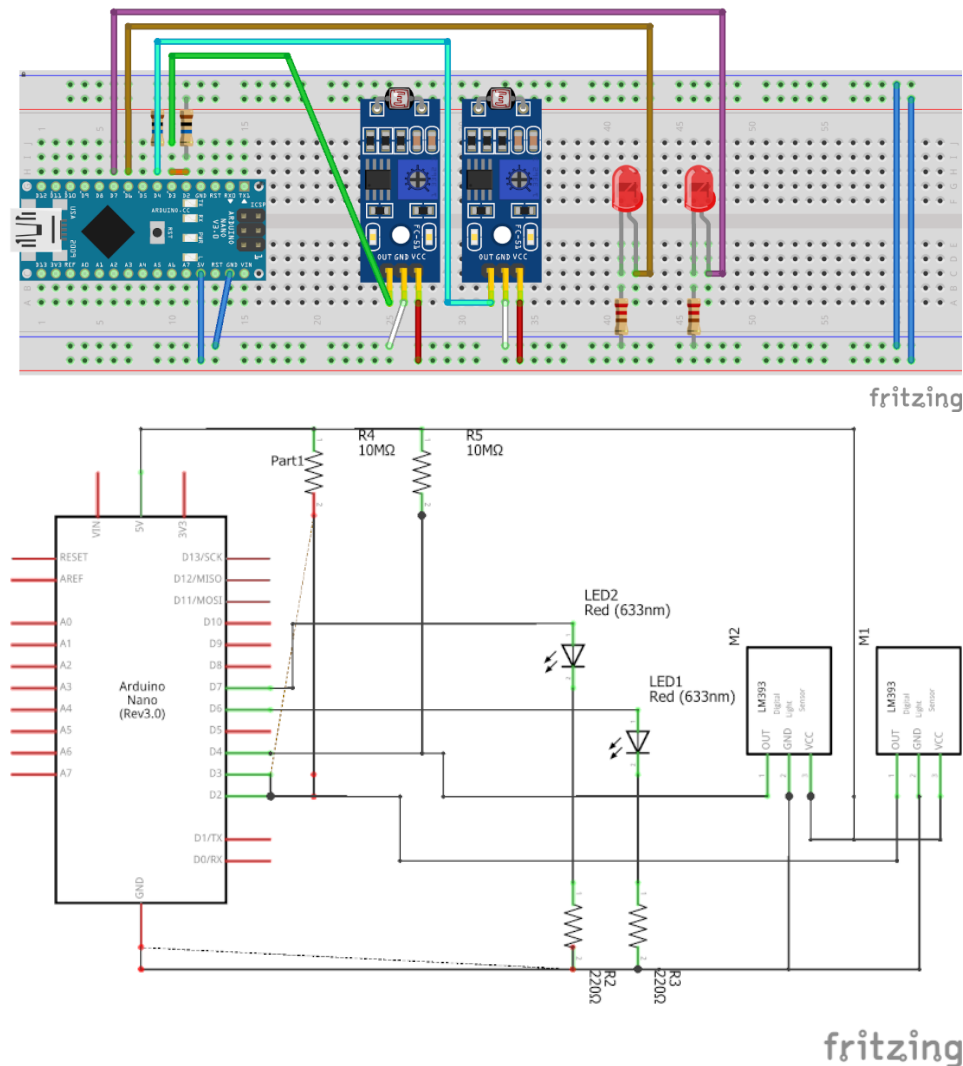
Das Entwickeln der Strategie für die Kommunikation, zu versuchen bereits bestehende Protokolle (als Basis) zu verwenden, war schnell erledigt. Der Versuch die nativen Protokolle des Arduinos zu verwenden scheiterte jedoch schnell, da alle Standardprotokolle viel zu schnell sind um mit LEDs verwendet werden zu können. Hinzukommt, dass uns zunächst entgangen ist, dass der LM392 das Signal negiert ausgibt, welches uns erst einiges später auffiel. Schließlich entschlossen wir uns für eine serielle Kommunikation auf Basis des Protokolls [I2C](#). Dieses Protokoll ist bei fast jedem modernen Microprocessor Hardwaremäßig implementiert für eine schneller Kommunikation. Bei diesem Protokoll ist in der Regel ein "Master" deklariert und mehrere "Slaves" welche alle gemeinsam an zwei Datenleitungen angeschlossen sind: "Data" und "Clock". Immer wenn die Spannung auf "Clock" von 0V auf 3.3/5V ändert wird auf "Data" der Zustand (Bit) abgelesen, bis sich

letztendlich, nach einigen Takten ein Byte (8 Bit) ergeben. Das Standardprotokoll arbeitet eigentlich mit Adressen um zu differenzieren an wen und von wem die Nachricht gesendet wurde.

Unser Protokoll übernimmt dabei nur das Prinzip von "Data" und "Clock", somit kann ein Empfangsmodul nur mit einem Sendermodul verbunden werden und vice versa.

## Unsere Hardware (Schematics)

Hier einmal die Schematics in der Arduino Schematic Software [fritzing](#)



In dieser Schematic wurde der zum Arduino Uno funktionsidentische Arduino Nano verwendet und die Grafik übersichtlicher und kompakter zu gestalten. Beide Arduinoversionen besitzen denselben Prozessor (ATmega328P) und dieselben Pins. Die einzigen Unterschiede beider sind zum einen der Formfaktor und zum anderen die Möglichkeit des Nanos, direkt auf ein Breadboard gesteckt werden zu können. Dargestellt wurde hier ein einzelner Arduino, der in der Lage ist sowohl zu Empfangen als auch zu Senden, wenn das passend aufgebaute Gegenstück existiert.

## Software

Unser Programm wurde in C++ für die Arduino IDE geschrieben, schaue man sich frühere Commits an (vor dem 18.12.2020) wurde noch für eine andere IDE geschrieben: PaltformIO welche zwar auf der Arduino IDE basiert, jedoch hier nur eine Extension für die IDE VS Code ist. Einfachheitshalber wurde jedoch entgültig zur Arduino IDE gewechselt.

LED\_SendSimple.ino

```
//init variables
int LED = 2;
int incomingByte, ByteToSend;
int SerialCounter = -1;
//serial input buffer used as a stack
int SerialBuffer[10];
int MaxValue = 4;
int ClockCounter = 0;
boolean state = 0;
int BitsToSend = -1;

void setup() {
  //init serial interface
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  //read serial input if available
  if(Serial.available() > 0){
    incomingByte = Serial.read();
    if (incomingByte >= 32){
      //increment serial counter
      SerialCounter++;
      //store recieved byte from serial into stack buffer
      SerialBuffer[SerialCounter] = incomingByte;
    }
  }
  //theoratically the clockcounter could be removed and the delay in the last line
  //set to 4*x
  ClockCounter++;
  if(ClockCounter >= MaxValue){
    ClockCounter = 0;
    state = !state;
    //if serial buffer is not empty
    if(SerialCounter > -1){
      //if current byte to send is not set
      if(BitsToSend <= -1){
        //set byte to send
        BitsToSend = SerialBuffer[SerialCounter];
        Serial.print("Sending byte: ");
        Serial.println(BitsToSend);
      }else{
        digitalWrite(LED, state);
        delay(70);
        //if led is on
        if(state){
          //decrease bit to send
          BitsToSend--;
          //if byte is sent, finish transmission
          if(BitsToSend <= 0){
            BitsToSend = -1;
          }
        }
      }
    }
  }
}
```

```

        SerialCounter--;
        //wait same time as in outer loop, to avoid flickering of last count
        delay(4*70);
        digitalWrite(LED, LOW);
        delay(5000);
    }
}
}
}
}
delay(70);
}

```

### LED\_RecieveSimple.ino

```

//init variables
int LDR = 2;
int ClockCounter = 0;
int FlashCounter = 0;
int MaxValue = 2500;
int HighFlanke = 0;
int Flash;
unsigned long long lastmillis = 0;
int skippedMillis = 0;

void setup()
{
    //init serial monitor
    Serial.begin(9600);
    //init pinmode
    pinMode(LDR, INPUT);
}

void loop()
{
    ClockCounter++;
    //read LDR value
    Flash = digitalRead(LDR);
    //if led is off
    if (Flash)
    {
        HighFlanke = 1;
    }
    //if LDR value changes from HIGH to LOW
    if (!Flash && HighFlanke)
    {
        HighFlanke = 0;
        //increment flashcounter
        //account for glitches (glitches most often split a flash into two flashes)
        long delta = millis() - lastmillis;
        if (delta >= 500 || skippedMillis >= 500)
        {

```

```

        //increment counter and reset other variables
        FlashCounter++;
        ClockCounter = 0;
        skippedMillis = 0;
    }
    else
    {
        //to avoid glitches to impact transmission
        skippedMillis += delta;
    }
    //debug messages
    Serial.print(FlashCounter);
    Serial.print(": ");
    Serial.print(delta);
    Serial.print("ms ");
    Serial.println(skippedMillis);
    lastmillis = millis();
}
//if transmission for one character finished
if (ClockCounter >= MaxValue && FlashCounter > 0)
{
    //write flashcounter as character
    Serial.write(FlashCounter);
    //write ascii value
    Serial.println(FlashCounter);
    //reset flashcounter for next character
    FlashCounter = 0;
}
else
    delay(1);
}

```

### Funktionsweise von Simple(Send/Recieve)

Die Funktionsweise von LED\_SendSimple.ino basiert auf einer seriellen Datenübertragung. Um ein Zeichen zu übertragen wird dazu der byte Zahlenwert übermittelt. Dazu bleibt die LED x Millisekunden an und x Millisekunden aus, in unserem fall jeweils ca. 350. Nach vollständigem Durchzählen eines Bytes wird die Übertragung dieses mit einer sechs sekündigen Wartezeit abgeschlossen.

Auf der Empfangsseite wird dabei auf ein eintreffendes Signal gewartet. Wenn mit einem Abstand von mindestens 500 ms ein Blinken erkannt wird, wird der derzeige Byte-Wert um eins erhöht, falls die Delta Zeit geringer ist, wird eine separate Variable erhöht. Durch das Verwenden der Deltazeit und dem Abfragen dieser zusätzliche Zeitvariable, wird der Einfluss von Glitches und Hazards bestmöglich minimiert. Bei einer Deltazeit von mehr als 2,5 Sekunden wird die Übertragung dieses Bytes abgeschlossen und dieser wird ausgegeben.

LightCom.ino (LED\_RecieveSynchron.ino + LED\_SendSynchron) (eigene Lösung)

```

//declare variables
//declare pins
int sender_clock = 5;
int sender_data = 6;

```

```
int reciever_clock = 2;
int reciever_clock_2 = 3;
int reciever_data = 4;

int incomingByte = 0;
long long last_millis = 0;
byte temp = 0;
int recieve_index = 7;
byte b = 0;
bool lastClockState = 0;

//declare timing
int ledDuration = 300;
int minSignalDur = 100;
int clDelay = 50;

void setup()
{
    //initialize Serial Monitor
    Serial.begin(9600);
    //set pin modes
    pinMode(sender_clock, OUTPUT);
    pinMode(sender_data, OUTPUT);
    pinMode(reciever_clock, INPUT);
    pinMode(reciever_clock_2, INPUT);
    pinMode(reciever_data, INPUT);
}

void loop()
{
    //check if serial queue contains items
    if (Serial.available() > 0)
    {
        //read incoming byte (characters)
        incomingByte = Serial.read();

        Serial.write(incomingByte);
        Serial.print(": ");
        Serial.print(incomingByte);
        Serial.print(" ");
        //send each bit seperately to serial monitor for debugging and to other
        arduino
        for (int i = 7; i >= 0; i--)
        {
            sendBit(bitRead(incomingByte, i));
        }
        Serial.println();
    }

    //check for changes on clock pin
    bool state = !digitalRead(reciever_clock);
    if (state != lastClockState)
    {
        lastClockState = state;
    }
}
```

```

        if (state == HIGH)
            clock_interrupt_start();
        else
            clock_interrupt_end();
    }
}

//read incoming bits
void clock_interrupt_start()
{
    bool dataval = !digitalRead(reciever_data);
    bitWrite(temp, recieve_index, dataval);
}

//verify incoming bit
void clock_interrupt_end()
{
    //calculate delta since clock fell, to check for hazards
    unsigned long delta = millis() - last_millis;
    //if delta > 100ms, it's nit a hazard or glitch from the photoresistor
    if (delta >= minSignalDur)
    {
        //write temporary byte to "longterm" byte
        b = temp;
        Serial.print(bitRead(b, recieve_index));
        //reduce recieve index to move to next bit
        recieve_index--;
        //if recieve_index < 0 we have successfully recieved a byte and print that to
the serial monitor
        if (recieve_index < 0)
        {
            Serial.write(b);
            //reset recieving variables to default state
            recieve_index = 7;
            b = 0x0;
        }
        //reset for delta caluclation
        last_millis = millis();
    }
    else
    {
        //if a glitch was detected, delete temporarily recieved bit
        temp = b;
    }
}

//send single bit
void sendBit(bool bitt)
{
    Serial.print(bitt);
    //change data pin
    digitalWrite(sender_data, bitt);
    delay(clDelay);
    //change clock to HIGH

```

```

digitalWrite(sender_clock, HIGH);
//wait to be able to differentiate between valid bit and a hazard
delay(ledDuration);
//change both pins to default state
digitalWrite(sender_clock, LOW);
delay(clDelay);
digitalWrite(sender_data, LOW);
delay(ledDuration);
}

```

## Funktionsweise von LightCom

Dieses Projekt ist für eine bidirektionale Kommunikation geeignet, d.h. beide Arduinos können miteinander kommunizieren. Die Kommunikation basiert auf dem simplen seriellen Protokoll namens I<sup>2</sup>C, bei dem zwei Verbindungen für eine serielle Datenübertragung verwendet werden. Diese Verbindungen heißen "Clock" und "Data" bzw. "SDA" (Serial Data) und "SCL" (Serial Clock). Bei der Modifikation dieses Protokolls wurde die Funktionsweise nicht verändert, jedoch musste das Verhalten des LDRs beachtet werden, da dieser Logisch negiert. Um nun einen Byte zu senden wird dieser Bitweise übertragen, dabei wird der MSB als erstes gesendet und die Übertragung eines Bytes endet mit dem LSB. Das Übermitteln eines einzelnen Bits erfolgt dabei wie folgt: Die "Data"-LED wird auf den jeweiligen Wert des Bits gesetzt (HIGH für 1 und LOW für 0), nach einem kurzen Delay, um sicherzustellen, dass die LED ihre gewünschte Helligkeit erreicht hat, wird die "Clock"-LED auf HIGH gesetzt, kurze Zeit später werden beide LEDs, unabhängig von ihrem vorherigen Zustand, wieder auf LOW gesetzt. Nach dem Empfangen von 8 Bits, bereits entsprechend dem MSB und LSB in einem Byte gespeichert, werden diese als ASCII Buchstabe auf dem Seriellen Monitor ausgegeben.

Link zum aktuellen Programmcode: [Github](#), [IDE3](#)

Link zum aufbereiteten/alternativen Code (eigenständig entwickelt von Tom Arlt): [Github](#), [IDE3](#)

## Verwendung

(Übersetzt und aufbereitet aus [Readme.md](#)) Um unser Projekt zu verwenden, müssen zwei Schaltungen aufgebaut sein, wobei mindestens zwei LEDs und zwei Empfänger aufgebaut sein müssen, um zumindest die Einseitige Kommunikation testen zu können. Zu beachten ist, dass jedes LED-LM392 Paar zu allen anderen Paaren abgeschirmt sein sollte, da es sonst zu Interferenzen kommen kann. Zudem muss das Potenziometer des LM392 eingestellt werden nur die dazugehörige eingeschaltete LED zu erkennen und nicht das Umgebungslicht. Bei der Erstellung der LED-LM392 Paare ist auf die Unterscheidung zwischen "Data" und "Clock" zu achten:

```

Arduino 1      <-> Arduino 2
sender_clock    <-> reciever_clock
sender_data     <-> reciever_data
reciever_clock  <-> sender_clock
reciever_data   <-> sender_data

```



Ist das Programm einmal auf beiden Arduinos hochgeladen, ist die Arduino IDE (o.ä.) nicht mehr notwendig. Nun kann über einen Seriellen Monitor, wie zum Beispiel der integrierte Monitor der [Arduino IDE](#) oder [Putty](#) über den Arduino kommuniziert werden. In diesen Tools muss noch der Port, über den der Arduino angeschlossen ist, angegeben werden, sowie die Baud Rate des Seriellen Monitor, welche hier **9600** beträgt.

## Reflexion

Wie schon in **Strategie** erwähnt, war die grundsätzliche Strategie schnell gefunden, nur das ausarbeiten hat, dank der Rückschläge, etwas länger gedauert. Nach den ersten Rückschlägen und das Festsetzen auf die entgeltliche Implementationsstrategie war auch das Entwickeln eines ersten Gerüsts relativ straight forward. Das Debuggen sowie ein ordentlicher Aufbau der Schaltung war dann die größere Herausforderung. Da durch ständig wechselnde Lichtverhältnisse der Photoresistor nicht immer wie gewünscht funktionierte, verzögerte sich das Programm-Debuggen ungemein. Erst nach vielen Stunden rumgrübelns, rumschrauben und ausprobieren, stellte sich als einzige zuverlässige Methode eine eigene Blackbox für jedes LED-LM392 Paar heraus.

## Zusammenfassung

Die Aufgaben waren, wenn auch leicht missverständlich organisiert, einfach zu bearbeiten. Wir hatten schnell eine grundsätzliche Strategie für die Programmentwicklung gefunden und hatten binnen einiger Stunden nach den Fehlversuchen einen ersten ordentlichen Prototypen stehen. Somit hatten wir die grundsätzliche Aufgabe "zwei Arduinos mithilfe von LEDs und LDRs kommunizieren zu lassen" gelöst. Währenddessen sich die Ersten bereits an die Dokumentation gesetzt hatten, haben sich andere im Internet auf die Suche gemacht ein angemessenes Schematic-Tool ausfindig zu machen und eine digitale Schaltung zu erstellen. Erst später, als alle gemeinsam an der Dokumentation arbeiteten, fiel auf, dass die Aufgaben der Anleitung zum Kommunikationssystem zu bearbeiten sind. Als die Dokumentation bereits in den letzten Zügen war, wurden die zusätzlichen Aufgaben fertiggestellt, dabei schafften wir es jedoch nicht die Kommunikation der Arduinos mithilfe dieser Aufgaben über die digitalen Kommunikationswege erfolgreich zu realisieren, da die Photoresistoren die LEDs nur unzureichend erkennen konnten. Zusätzlich kamen noch Timingproblemen hinzu, die durch die Kommunikation entstanden sind. Insgesamt hat die Gruppe gut zusammengearbeitet und ist recht zügig zu einer ersten Lösung gekommen.