

# Versuch V2

C752 Digitaltechnik

*Abnahme: 16. Januar 2024*

Stand: 2. Januar 2024

*Tom Mohr*

*Martin Ohmeyer*

# Inhaltsverzeichnis

<b>1</b>	<b>Aufgabe 3</b>	<b>1</b>
1.1	Aufgabe 3.1 . . . . .	1
<b>2</b>	<b>Aufgabe 4</b>	<b>2</b>
2.1	Aufgabe 4.2 . . . . .	2
2.2	Aufgabe 4.4 . . . . .	2
2.3	Aufgabe 4.2 . . . . .	9

# 1 Aufgabe 3

## 1.1 Aufgabe 3.1

**Aufgabenstellung** Sollten Sie bei den folgenden Punkten Wissenslücken feststellen, füllen Sie diese bitte vor dem Laborversuch selbstständig auf z.B. durch YouTube Videos.

**Vorüberlegung** Es bestehen keine größeren Defizite in den Gebieten *RS232*, *DMX*, *ASCII* und *Baudrate/Bitrate*.

**Schlussfolgerung** Der Wissensstand ist ausreichend, um die folgenden Aufgaben gewissenhaft erledigen zu können.

## 2 Aufgabe 4

### 2.1 Aufgabe 4.2

**Vorteile der differentiellen Signalübertragung** Die differentielle Signalübertragung wird in allen modernen Protokollen verwendet. Fast alle Bussysteme, die außerhalb eines Gerätes liegen, greifen auf sie zurück. Ihre Stärke liegt in einer hohen Fehlerresistenz auch bei niedrigen Spannungen, was schnelle Übertragungsraten ermöglicht. Die Übertragung eines differenziellen Signals erfolgt dazu über zwei Kabel. Während das eine Kabel positive Spannungsausschläge verwendet, überträgt das andere Kabel negative Spannungsausschläge des gleichen Betrages. Das ursprüngliche Signal wird dann durch Subtraktion der beiden einzelnen Spannungen errechnet. Der große Vorteil: Verdrillt man die beiden Kabel, so wirkt eine Störung von außen auf beide gleichermaßen. Zwar ändern sich die Spannungsausschläge, die durch die jeweiligen Kabel übertragen werden, ihre Differenz bleibt jedoch unberührt und die übermittelten Daten unbeschädigt.

### 2.2 Aufgabe 4.4

**Aufgabenstellung** Wählen Sie ein Musikstück aus und programmieren Sie für die ersten 30s die Beleuchtungssequenz.

**Vorüberlegung** Um die DMX-Geräte ansteuern zu können, wird eine Klasse bereitgestellt, welche man mit einer einfachen Funktion, der man Adresse und gewünschten DMX-Werte gibt, bedienen kann. Da die Geräte nur an einem Ort erreichbar sind, sollte das Programm während der Entwicklung auch mit lokalen LEDs an den Pins des Arduinos ausführbar sein und einen schnellen Wechsel zwischen Test- & Produktionsmodus ermöglichen, ohne den Code in großem Umfang abändern zu müssen. Um dies zu realisieren, werden Klassen mit Polymorphismus benötigt. Da die Geräte ihre Befehle selbstständig abarbeiten sollen, kann hier nicht mit der Delay-Funktion des Arduinos gearbeitet werden. Ein anderer Ansatz ist, fortlaufend die verstrichene Zeit ermitteln und die Objekte selbst entscheiden lassen, wann sie ihre Befehle anführen.

**Durchführung** In der Hauptdatei werden die Geräte als Objekte initialisiert, das DMX-Interface vorbereitet und in einer Endlosschleife die Objekte aufgerufen um ihre Befehle abzuarbeiten.

## 2 Aufgabe 4

```
1  #define DEMO 0           //Demomodus AUS bei 0
2  #define BPM_IN_MS 484   //Quotient aus 60.000ms und BPM
3
4  #include <Arduino.h>
5  #include "RgbwSpotlight8Ch.h"
6  #include "MiniMovingHead14Ch.h"
7
8  //Geräte aus Klassen mit DMX Adresse instanzieren
9  #if DEMO == 0
10  /// @brief Enthält alle Spotlichter.
11  RgbwSpotlight8Ch spotlights[2] = {
12      {RgbwSpotlight8Ch(1)},
13      {RgbwSpotlight8Ch(37)}
14  };
15  /// @brief Enthält alle Movingheads.
16  MiniMovingHead14Ch movingHeads[2] = {
17      {MiniMovingHead14Ch(9)},
18      {MiniMovingHead14Ch(23)}
19  };
20
21  //Geräte aus Klassen mit Pin-Nummern instanzieren
22  #else
23
24  #include "RgbwSpotlight8ChDemo.h"
25  #include "MiniMovingHead14ChDemo.h"
26
27  /// @brief Enthält alle Spotlichter als LED.
28  RgbwSpotlight8ChDemo spotlights[2] = {
29      {RgbwSpotlight8ChDemo(2)},
30      {RgbwSpotlight8ChDemo(4)}
31  };
32  /// @brief Enthält alle Movingheads als LED.
33  MiniMovingHead14ChDemo movingHeads[2] = {
34      {MiniMovingHead14ChDemo(3)},
35      {MiniMovingHead14ChDemo(5)}
36  };
37  #endif
38
39  /// @brief Zeitstempel des letzten erkannten Beats.
40  unsigned int previousMillis = 0;
41
42  /// @brief Zeitstempel des Starts der Lichtshow.
43  unsigned int loopStartMillis = 0;
44
45  /// @brief Zeitintervalls, in denen erkannte Beats verarbeitet werden sollen.
46  unsigned int beatIntervals[1][2] = {
47      {4770, 31000},
48  };
49
50  /// @brief Prüft bei jedem Aufruf, ob inder bereits verstrichenen Zeit ein Beat
51  ↪ auftrat.
52  /// @param currentMillis Der Zeitpunkt des Aufrufs in.
53  void BeatDetector(unsigned int currentMillis) {
54      if ((currentMillis - previousMillis) >= BPM_IN_MS) {
55          previousMillis = currentMillis;
56
57          for (auto const &interval: beatIntervals) {
58              if (interval[0] <= currentMillis && currentMillis <= interval[1]) {
59                  //Event eines Beats
60                  for (auto &spotlight: spotlights) {
61                      spotlight.StartBlink(currentMillis, 255);
62                  }
63              }
64          }
65      }
66  }
```

## 2 Aufgabe 4

```
62         break;
63     }
64 }
65 }
66 }
67
68 /// @brief Bereitet den Programmablauf vor.
69 void setup() {
70     /// DMX Interface vorbereiten.
71     #if DEMO == 0
72         DmxSimpleClass::usePin(3);
73         DmxSimpleClass::maxChannel(44);
74     #endif
75
76     /// Start des Programms in der Ausgabe ankündigen.
77     Serial.begin(115200);
78     Serial.println(3);
79     delay(800);
80     Serial.println(2);
81     delay(800);
82     Serial.println(1);
83     delay(800);
84     Serial.println("GO");
85     delay(100);
86
87     loopStartMillis = millis();
88 }
89
90 /// @brief Speichert den aktuellen Zeitpunkt und gibt ihn an die Geräte zum
91 ↳ Ausführen von Befehlen weiter.
92 void loop() {
93     unsigned int currentMillis = millis() - loopStartMillis;
94     BeatDetector(currentMillis);
95
96     for (auto &spotlight: spotlights) {
97         spotlight.CleanUp(currentMillis);
98         spotlight.RunTick(currentMillis);
99     }
100     for (auto &movingHead: movingHeads) {
101         movingHead.CleanUp(currentMillis);
102         movingHead.RunTick(currentMillis);
103     }
104 }
```

Damit die Geräte später einheitlich angesteuert werden können und eine gleiche Befehlsstruktur verwenden, definiert *DmxCommand* die Befehle. Jeder Befehl enthält einen Zeitstempel, den Kanal und welchen Wert dieser annehmen soll.

```
1  #ifndef DmxCommand_h
2  #define DmxCommand_h
3
4  /// @brief Beschreibt einen Befehl, der über DMX ausgeführt werden soll.
5  struct DmxCommand {
6      /// @brief Der Zeitpunkt der geplanten Ausführung.
7      unsigned int executionTime;
8      /// @brief Der Kanal oder die Funktion, welche bedient wird.
9      int function;
10     /// @brief Der Wert, welcher die Funktion verändert.
11     unsigned char value;
12 };
13 #endif
```

## 2 Aufgabe 4

Um dem Polymorphismus gerecht zu werden, wird eine Basisklasse benötigt. Von *DmxDevice* erben die Klassen für Scheinwerfer und Movingheads mit einer Grundstruktur von Funktionen und Variablen.

```
1  #ifndef DmxDevice_h
2  #define DmxDevice_h
3
4  #include "DmxSimple.h"
5  #include "DmxCommand.h"
6  #include <Arduino.h>
7
8  /// @brief Basisklasse für alle DMX-Geräte
9  class DmxDevice {
10 public:
11     explicit DmxDevice(unsigned short address) : Address(address) {}
12
13     /// @brief Die Adresse des Geräts
14     unsigned short Address;
15
16     /// @brief Erzeugt ein DMX-Signal an einem Kanal des Geräts.
17     /// @param channel Der Kanal des Geräts, beginnend bei Kanal 0.
18     /// @param value Der Wert des Signals im Bereich von 0 bis 255.
19     virtual void Set(int channel, unsigned char value) {
20         DmxSimpleClass::write(static_cast<int>(Address + channel), value);
21     };
22
23     /// @brief Verarbeitet Befehle zu ihren Zeitpunkten.
24     /// @param currentMillis Der aktuelle Zeitstempel.
25     virtual void RunTick(unsigned int currentMillis) = 0;
26
27     /// @brief Bereinigt Spuren vergangener Befehle oder führt diese fort, sofern
28     ↪ sie mehrere Aufrufe benötigen.
29     /// @param currentMillis Der aktuelle Zeitstempel.
30     virtual void CleanUp(unsigned int currentMillis) = 0;
31
32     /// @brief Werkzeug für das debuggen von Befehlen.
33     /// @param currentMillis Der aktuelle Zeitstempel.
34     static void PrintTimeToSerial(unsigned int currentMillis) {
35         Serial.print("Time elapsed: ");
36         Serial.print(currentMillis);
37         Serial.println("ms");
38     }
39 };
40 #endif
```

Von dieser Basisklasse kann nun die Klasse *RgbwSpotlight8Ch* für die Spotlichter erben, ihre Befehle definieren und Funktionen erweitern.

```
1  #ifndef RgbwSpotlight8Ch_h
2  #define RgbwSpotlight8Ch_h
3
4  #include "DmxDevice.h"
5  #include "DmxCommand.h"
6
7  /// @brief Beschreibt einen DMX-Scheinwerfer mit acht Kanälen.
8  class RgbwSpotlight8Ch : public DmxDevice {
9 public:
10     explicit RgbwSpotlight8Ch(unsigned short address) : DmxDevice(address) {}
11
12     /// @brief Alle Funktionen, welche das Gerät unterstützt.
```

## 2 Aufgabe 4

```
13  enum Functions {
14      TotalDimming,
15      RedDimming,
16      GreenDimming,
17      BlueDimming,
18      WhiteDimming,
19      TotalStrobe,
20      FuncSelection,
21      FuncSpeed,
22
23      Blink = 100,
24      Fade = 101,
25      BlinkTimeout = 200,
26      FadeTimeout = 201,
27
28      Stop = 300,
29      PrintTime = 301
30  };
31  bool blinkOn = false;
32  bool isFading = false;
33
34  /// @brief List der Befehle des Geräts.
35  DmxCommand commandList[24] = {
36      {100,  BlueDimming,  255},
37      {200,  TotalDimming, 100},
38      {400,  Blink,        255},
39      {800,  Blink,        255},
40      {1100, Blink,        255},
41      {1580, Blink,        255},
42      {1950, Blink,        255},
43      {2350, Blink,        255},
44      {2700, Blink,        255},
45      {2990, BlinkTimeout, 255},
46      {3000, Blink,        255},
47      {3400, BlinkTimeout, 100},
48      {3550, Blink,        255},
49      {3900, Blink,        255},
50      {4200, Blink,        255},
51      {4700, TotalDimming, 0},
52      {4701, RedDimming,   100},
53      {16000, RedDimming,  255},
54      {16001, BlueDimming, 0},
55      {22000, GreenDimming, 255},
56      {22001, RedDimming,  0},
57      {26000, WhiteDimming, 200},
58      {31000, Fade,        255},
59      {32800, Stop,        0}
60  };
61
62  void RunTick(unsigned int currentMillis) override {
63      DmxCommand cmd = commandList[commandIndex];
64      if (cmd.executionTime < currentMillis) {
65          if (cmd.function != Stop) {
66              if (cmd.function < 200) {
67                  cmd.value = static_cast<unsigned char>(cmd.value);
68              }
69
70              switch (cmd.function) {
71                  case PrintTime:
72                      PrintTimeToSerial(currentMillis);
73                  case Blink:
74                      StartBlink(currentMillis, cmd.value);
```



## 2 Aufgabe 4

```
75         break;
76     case Fade:
77         StartFading(currentMillis);
78     case BlinkTimeout:
79         blinkTimeout = cmd.value;
80     case FadeTimeout:
81         fadingTimeout = cmd.value;
82     case TotalDimming:
83         totalDimmingValue = cmd.value;
84         Set(TotalDimming, totalDimmingValue);
85     default:
86         Set(static_cast<Functions>(cmd.function), cmd.value);
87         break;
88     }
89     commandIndex++;
90 } else {
91     Set(TotalDimming, 0);
92 }
93 }
94 };
95
96 void Cleanup(unsigned int currentMillis) override {
97     // Blinken beenden
98     if (blinkOn && (currentMillis - blinkStart) > blinkTimeout) {
99         Set(TotalDimming, totalDimmingValue);
100         blinkOn = false;
101     }
102
103     // Dimmen fortsetzen falls aktiv
104     if (isFading && (currentMillis - fadingLastCall) > fadingTimeout) {
105         totalDimmingValue--;
106         fadingLastCall = currentMillis;
107         Set(TotalDimming, totalDimmingValue);
108         isFading = fadingInterval[1] < totalDimmingValue;
109     }
110 }
111
112 /// @brief Bereitet das Blinken des Lichts vor.
113 /// @param currentMillis Der aktuelle Zeitstempel.
114 /// @param value Die Stärke des blinkenden Lichts.
115 void StartBlink(unsigned int currentMillis, unsigned char value) {
116     blinkOn = true;
117     blinkStart = currentMillis;
118     Set(TotalDimming, value);
119 }
120
121 /// @brief Bereitet das dimmen des Lichts vor.
122 /// @param currentMillis Der aktuelle Zeitstempel.
123 void StartFading(unsigned int currentMillis) {
124     isFading = true;
125     fadingLastCall = currentMillis;
126     totalDimmingValue = fadingInterval[0];
127     Set(TotalDimming, totalDimmingValue);
128 }
129
130 protected:
131 /// @brief Die aktuelle Zeile in der Befehlsliste.
132 unsigned short commandIndex = 0;
133 /// @brief Die aktuelle Gesamthelligkeit.
134 unsigned char totalDimmingValue = 0;
135
136 /// @brief Zeitpunkt des Starts des blinkens.
```

## 2 Aufgabe 4

```
137     unsigned int blinkStart = 0;
138     /// @brief Dauer des blinkenden Lichts.
139     unsigned short blinkTimeout = 100;
140
141     /// @brief Letzter Zeitpunkt des dimmens.
142     unsigned int fadingLastCall;
143     /// @brief Zeit zwischen dem Dimmen zweier Werte.
144     unsigned char fadingTimeout = 7;
145     /// @brief Intervall des dimmens.
146     unsigned char fadingInterval[2] = {
147         255,    ///Anfang
148         0       ///Ende
149     };
150 };
151
152 #endif
```

Damit im Testmodus statt den Scheinwerfern LEDs verwendet werden können, erbt eine weitere Klasse *RgbwSpotlight8ChDemo* von *RgbwSpotlight8Ch*, um das DMX-Interface lokal emulieren zu können.

```
1  #ifndef RgbwSpotlight8ChDemo_h
2  #define RgbwSpotlight8ChDemo_h
3
4  #include "RgbwSpotlight8Ch.h"
5
6  /// @brief Beschreibt die Emulation eines DMX-Scheinwerfer mit acht Kanälen.
7  class RgbwSpotlight8ChDemo : public RgbwSpotlight8Ch {
8  public:
9      explicit RgbwSpotlight8ChDemo(unsigned short address) :
10         ↳ RgbwSpotlight8Ch(address) {
11         pinMode(Address, OUTPUT);
12     }
13
14     /// @brief Überschreibt den DMX-Ausgang mit lokalen Pins.
15     /// @param channel Die Funktion oder der Kanal.
16     /// @param value Der Wert der Funktion
17     void Set(int channel, unsigned char value) override {
18         switch (channel) {
19             case 0:
20                 if (value <= 100) {
21                     digitalWrite(Address, LOW);
22                 } else {
23                     digitalWrite(Address, HIGH);
24                 }
25                 break;
26             default:
27                 break;
28         }
29     }
30
31     /// @brief Überschreibt den Reinigungsprozess um Pins benutzen zu können.
32     /// @param currentMillis Der aktuelle Zeitstempel.
33     void CleanUp(unsigned int currentMillis) override {
34         if (blinkOn && (currentMillis - blinkStart) > 100) {
35             digitalWrite(Address, LOW);
36             blinkOn = false;
37         }
38     };
39
40 #endif
```

## 2 Aufgabe 4

**Schlussfolgerung** Zur Kompilierzeit wird festgelegt, ob sich das Programm im Test- oder Produktionsmodus befindet. Objekte aus Klassen können in beiden Modi gleich angesteuert werden, was das testen von Befehlen stark vereinfacht. Das Programm lässt die Geräte in einer Endlosschleife selbstständig ihre Befehle abarbeiten, ohne dass diese sich durch Verzögerungen blockieren. Der Versuch ist erfolgreich.

### 2.3 Aufgabe 4.5

**Aufgabenstellung** Bitte räumen Sie auf und setzen Sie ggf. veränderte Arduinos zurück.

**Durchführung** Mithilfe dieser Befehle werden die Arduinos zurückgesetzt:

```
$ /mnt/datadisk/reset_arduino.sh dmx ttyUSB1
```

```
$ /mnt/datadisk/reset_arduino.sh rs232 ttyUSB1
```

**Schlussfolgerung** Die Arduinos befinden sich in ihrem Ausgangszustand und wurden Ordnungsgemäß zurück geräumt.