Versuch V2

C752 Digitaltechnik

Abnahme: 16. Januar 2024
Stand: 16. Januar 2024

Tom Mohr
Martin Ohmeyer

Inhaltsverzeichnis

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 4.1

Aufgabenstellung Schließen Sie das Oszilloskop an den TXD Ausgang des Arduinos an. Analysieren Sie das Signal.

Vorbereitung Um den Arduino verwenden zu können, benötigt er eine Betriebsspannung von 5V, welche das Netzteil bereitstellt. Der Arduino wird mit den Pins VIN und GND mit dem Netzteil verbunden.

Durchführung Mit einem Trigger wird die Erfassung des Signals am Oszilloskop angehalten, damit es abgelesen werden kann.

Verwendetes Protokoll:	UART
Baudrate:	8333 Bd
Anzahl Stopbits:	1
Fehler:	Keine Paritätsbits
Daten:	Das

Schlussfolgerung: Verwendetes Protokoll Bei dem verwendeten Protokoll gibt es (der Aufgabenstellung nach) zwei Möglichkeiten: DMX und RS232. Während DMX differentiell übertragen wird, ist ist dies bei RS232 nicht der Fall. In dieser Aufgabe ist das Signal nicht differentiell, weswegen es sich um **RS232** handeln muss.

Schlussfolgerung: Baudrate
geg.: $f_s = 120\mu s = 1,2 \times 10^{-4} s$ **ges.:** T_s

Lsg:

$$T_s = \frac{1s}{f_s}$$
$$T_s = \frac{1s}{1,2 \times 10^{-4} s}$$
$$T_s = 8333.3\,\text{Bd} \approx 8.3\,\text{kBd}$$

Antwort: Die Baudrate beträgt rund 8.3 kBd.

Schlussfolgerung: Stopbits Es wird ein Stopbit verwendet. Dieses ist 1.

Schlussfolgerung: Fehler bei der Übertragung Um Fehler bei der Übertragung von Daten festzustellen, gibt es drei Möglichkeiten: Die Verwendung eines Paritätsbits, die Verwendung einer Prüfsumme oder die Anwendung eines kryptografischen Verfahrens. Diese Möglichkeiten werden, in Reihenfolge ihrer Auflistung, immer sicherer, aber auch komplexer. Im Fall des während des Experimentes übertragenen Signals, kam keine Fehlerüberprüfung zum Einsatz. Anhand des Signals selbst ist folglich nicht feststellbar, ob Fehler bei der Übertragung auftraten. Nach der Auswertung der Daten ist jedoch ersichtlich: Alle Daten wurden korrekt übertragen.

Schlussfolgerung: Übertragene Daten Bei der Auswertung des Signal ist zu beachten, dass das *least significant bit* zuerst und das *most significant bit* zuletzt übertragen wird. Das Signal muss also "rückwärts"gelesen werden. Zur Interpretation der erhaltenen Dezimalzahlen wird die ASCII-Tabelle herangezogen. Es folgt die Analyse der ersten 30 bits des übertragenen Signals. Startbits sind grün, Stopbits rot markiert.

$$0\,\underbrace{00100010}_{68\,\equiv\,D}\,10\,\underbrace{10000110}_{97\,\equiv\,a}\,10\,\underbrace{11001110}_{115\,\equiv\,s}\,1$$

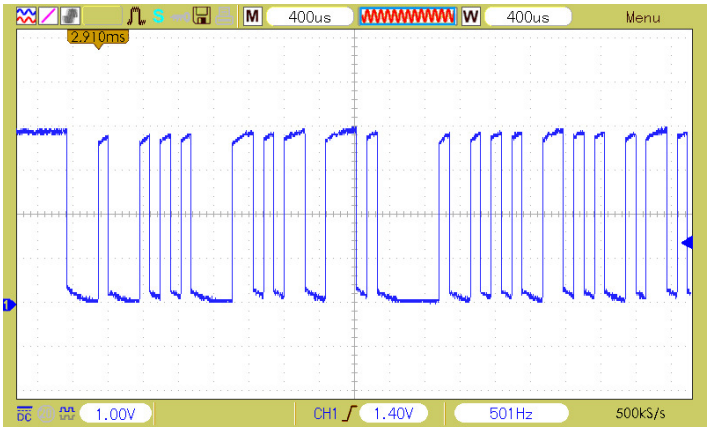

Abb. 2.1: Bildschirmfoto

2.2 Aufgabe 4.2

Aufgabenstellung Schließen Sie das Oszilloskop an die D+ und D- Pins des DMX-Boards an. Nutzen Sie die Mathematikfunktion um das Differenzsignal darzustellen.

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 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, welche durch die jeweiligen Kabel übertragen werden, ihre Differenz bleibt jedoch unberührt und die übermittelten Daten unbeschädigt.

Durchführung Die Baudrate beträgt 31 205 Bd.

Kanal	Binär	Dezimal	Parameter
1	00000000	255	
2	00000000	255	
3	00000000	255	
4	00000000	255	
5	00000000	255	
6	00000000	255	
7	00000000	255	
8	00000000	255	

Schlussfolgerung

2.3 Aufgabe 4.3

Aufgabenstellung Benutzen Sie den Logicanalyser um das Zitat zu dekodieren welches der RS232-Arduino sendet.

Vorbereitung Um das Signal mittels des Logicanalyser auslesen zu können, muss dieser über einen beliebigen Kanal und GND mit dem Arduino verbunden werden. In der Anwendung PulseView wird für die Darstellung der Decoder DMX512 ausgewählt und mit dem angeschlossenen Kanal verknüpft.

Durchführung

Schlussfolgerung

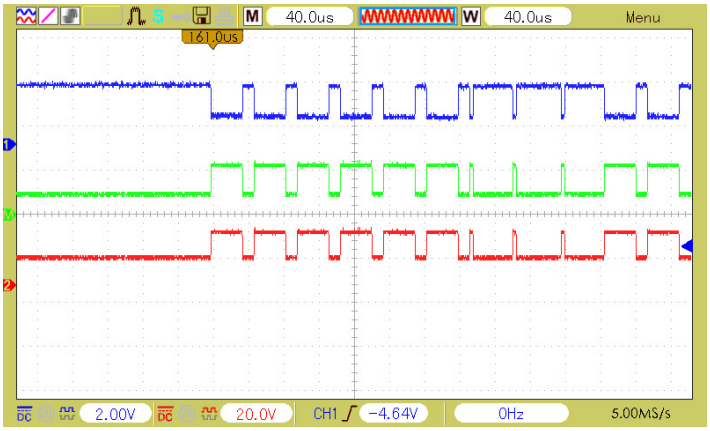

Abb. 2.2: Bildschirmfoto

2.4 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.

```
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  #include "CommandList.h"
8
9  //Geräte aus Klassen mit DMX Adresse instanziieren
10 #if DEMO == 0
11 /// @brief Enthält alle Spotlichter.
12 RgbwSpotlight8Ch spotlights[2] = {
13         {RgbwSpotlight8Ch(1)},
```

```
14          {RgbwSpotlight8Ch(37)}
15  };
16  /// @brief Enthält alle Moving heads.
17  MiniMovingHead14Ch movingHeads[2] = {
18          {MiniMovingHead14Ch(9)},
19          {MiniMovingHead14Ch(23)}
20  };
21
22  //Geräte aus Klassen mit Pin-Nummern instanziieren
23  #else
24
25  #include "RgbwSpotlight8ChDemo.h"
26  #include "MiniMovingHead14ChDemo.h"
27
28  /// @brief Enthält alle Spotlichter als LED.
29  RgbwSpotlight8ChDemo spotlights[2] = {
30          {RgbwSpotlight8ChDemo(2)},
31          {RgbwSpotlight8ChDemo(4)}
32  };
33  /// @brief Enthält alle Movingheads als LED.
34  MiniMovingHead14ChDemo movingHeads[2] = {
35          {MiniMovingHead14ChDemo(3)},
36          {MiniMovingHead14ChDemo(5)}
37  };
38  #endif
39
40  /// @brief Die aktuelle Zeile in der Befehlsliste.
41  unsigned short dmxCommandIndex = 0;
42
43  /// @brief Zeitstempel des letzten erkannten Beats.
44  uint16_t previousMillis = 0;
45
46  /// @brief Zeitstempel des Starts der Lichtshow.
47  uint16_t loopStartMillis = 0;
48
49  /// @brief Zeitintervalls, in denen erkannte Beats verarbeitet werden sollen.
50  unsigned short beatIntervals[1][2] = {
51          {4770, 30000},
52  };
53
54  /// @brief Prüft bei jedem Aufruf, ob inder bereits verstrichenen Zeit ein Beat
    ↪  auftrat.
55  /// @param currentMillis Der Zeitpunkt des Aufrufs in.
56  void BeatDetector(uint16_t currentMillis)
57  {
58      if ((currentMillis - previousMillis) >= BPM_IN_MS)
59      {
60          previousMillis = currentMillis;
61
62          for (auto const *interval: beatIntervals)
63              if (interval[0] <= currentMillis && currentMillis <= interval[1])
64              {
65                  //Event eines Beats
66                  for (auto &spotlight: spotlights)
67                  {
68                      spotlight.StartBlink(currentMillis, 255);
69                  }
70                  break;
71              }
72
73      }
74  }
```

```
75
76  /// @brief Bereitet den Programmablauf vor.
77  void setup()
78  {
79  #if DEMO == 0
80      // DMX Interface vorbereiten.
81      DmxSimpleClass::usePin(3);
82      DmxSimpleClass::maxChannel(44);
83  #endif
84
85      // Start des Programms in der Ausgabe ankündigen.
86      Serial.begin(115200);
87      Serial.println(3);
88      delay(800);
89      Serial.println(2);
90      delay(800);
91      Serial.println(1);
92      delay(800);
93      Serial.println("GO");
94      delay(100);
95
96      loopStartMillis = millis();
97  }
98
99  /// @brief Speichert den aktuellen Zeitpunkt und gibt ihn an die Geräte zum
    ↪  Ausführen von Befehlen weiter.
100 void loop()
101 {
102     uint16_t currentMillis = millis() - loopStartMillis;
103     BeatDetector(currentMillis);
104
105     DmxCommand currentCommand = dmxCommandList[dmxCommandIndex];
106     if (currentCommand.executionTime < currentMillis)
107     {
108         switch (currentCommand.deviceGroup)
109         {
110             case 0:
111                 for (auto &spotlight: spotlights)
112                 {
113                     spotlight.RunTick(currentMillis, currentCommand);
114                 }
115                 break;
116             case 1:
117                 for (auto &movingHead: movingHeads)
118                 {
119                     movingHead.RunTick(currentMillis, currentCommand);
120                 }
121                 break;
122             default:
123                 Serial.print(&"Unknown device group: "[currentCommand.deviceGroup]);
124                 Serial.println(&" at "[currentMillis]);
125                 break;
126         }
127         dmxCommandIndex++;
128     }
129
130     for (auto &spotlight: spotlights)
131     {
132         spotlight.CleanUp(currentMillis);
133     }
134     for (auto &movingHead: movingHeads)
135     {
```

```
136             movingHead.CleanUp(currentMillis);
137         }
138     }
```

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 Die Gerätegruppe, welche verwendet werden soll.
7       uint8_t deviceGroup;
8       /// @brief Der Zeitpunkt der geplanten Ausführung.
9       unsigned short executionTime;
10      /// @brief Der Kanal oder die Funktion, welche bedient wird.
11      short function;
12      /// @brief Der Wert, welcher die Funktion verändert.
13      uint32_t value;
14  };
15  #endif
```

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  {
11  public:
12      explicit DmxDevice(unsigned short address) : Address(address)
13      { }
14
15      /// @brief Die Adresse des Geräts
16      unsigned short Address;
17
18      /// @brief Erzeugt ein DMX-Signal an einem Kanal des Geräts.
19      /// @param channel Der Kanal des Geräts, beginnend bei Kanal 0.
20      /// @param value Der Wert des Signals im Bereich von 0 bis 255.
21      virtual void Set(unsigned short channel, uint8_t value)
22      {
23          DmxSimpleClass::write(static_cast<int>(Address + channel), value);
24      };
25
26      /// @brief Verarbeitet Befehle zu ihren Zeitpunkten.
27      /// @param currentMillis Der aktuelle Zeitstempel.
28      virtual void RunTick(uint16_t currentMillis, DmxCommand cmd) = 0;
29
30      /// @brief Bereinigt Spuren vergangener Befehle oder führt diese fort, sofern
        ↪  sie mehrere Aufrufe benötigen.
31      /// @param currentMillis Der aktuelle Zeitstempel.
```

```
32      virtual void CleanUp(uint16_t currentMillis) = 0;
33
34      /// @brief Werkzeug für das debuggen von Befehlen.
35      /// @param currentMillis Der aktuelle Zeitstempel.
36      static void PrintTimeToSerial(uint16_t currentMillis)
37      {
38          Serial.print("Time elapsed: ");
39          Serial.print(currentMillis);
40          Serial.println("ms");
41      }
42
43  protected:
44      uint8_t red = 0, green = 0, blue = 0;
45
46      /// \brief Konvertiert eine Dezimalzahl in einzelne RGB-Werte
47      /// \param rgbw Die Dezimalzahl
48      /// \param red Rot Wert
49      /// \param green Grün Wert
50      /// \param blue Blau Wert
51      static void ConvertDecimalToRgb(uint32_t rgbw, uint8_t &red, uint8_t &green,
        ↪  uint8_t &blue)
52      {
53          red = rgbw >> 16 & 0xFF;
54          green = rgbw >> 8 & 0xFF;
55          blue = rgbw & 0xFF;
56      }
57  };
58
59  #endif
```

Von dieser Basisklasse kann nun die Klasse RgbwSpotlight8Ch für die Spotlichter erben 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   {
10  public:
11      explicit RgbwSpotlight8Ch(unsigned short address) : DmxDevice(address)
12      { }
13
14      /// @brief Alle Funktionen, welche das Gerät unterstützt.
15      enum Functions
16      {
17          TotalDimming,
18          RedDimming,
19          GreenDimming,
20          BlueDimming,
21          WhiteDimming,
22          TotalStrobe,
23          FuncSelection,
24          FuncSpeed,
25
26          Blink = 100,
27          Fade = 101,
28          RGB = 102,
29          BlinkTimeout = 200,
```

```
30
31          Stop = 300,
32          PrintTime = 301
33      };
34      bool blinkOn = false;
35      bool isFading = false;
36
37      void RunTick(uint16_t currentMillis, DmxCommand cmd) override
38      {
39          if (cmd.function != Stop)
40          {
41              switch (cmd.function)
42              {
43                  case PrintTime:
44                      PrintTimeToSerial(currentMillis);
45                  case Blink:
46                      StartBlink(currentMillis, cmd.value);
47                      break;
48                  case Fade:
49                      fadingTimeout = cmd.value;
50                      StartFading(currentMillis);
51                  case BlinkTimeout:
52                      blinkTimeout = cmd.value;
53                  case TotalDimming:
54                      totalDimmingValue = cmd.value;
55                      Set(TotalDimming, totalDimmingValue);
56                  case RGB:
57                      ConvertDecimalToRgb(cmd.value, red, green, blue);
58                      Set(RedDimming, red);
59                      Set(GreenDimming, green);
60                      Set(BlueDimming, blue);
61                      break;
62                  default:
63                      Set(static_cast<Functions>(cmd.function), cmd.value);
64                      break;
65              }
66          } else
67          {
68              Set(TotalDimming, 0);
69          }
70      };
71
72      void CleanUp(uint16_t currentMillis) override
73      {
74          // Blinken beenden
75          if (blinkOn && (currentMillis - blinkStart) > blinkTimeout)
76          {
77              Set(TotalDimming, totalDimmingValue);
78              blinkOn = false;
79          }
80
81          // Dimmen fortsetzen falls aktiv
82          if (isFading && (currentMillis - fadingLastCall) > fadingTimeout)
83          {
84              totalDimmingValue--;
85              fadingLastCall = currentMillis;
86              Set(TotalDimming, totalDimmingValue);
87              isFading = fadingInterval[1] < totalDimmingValue;
88          }
89      }
90
91      /// @brief Bereitet das Blinken des Lichts vor.
```

```
92      /// @param currentMillis Der aktuelle Zeitstempel.
93      /// @param value Die Stärke des blinkenden Lichts.
94      void StartBlink(uint16_t currentMillis, uint8_t value)
95      {
96          blinkOn = true;
97          blinkStart = currentMillis;
98          Set(TotalDimming, value);
99      }
100
101     /// @brief Bereitet das dimmen des Lichts vor.
102     /// @param currentMillis Der aktuelle Zeitstempel.
103     void StartFading(uint16_t currentMillis)
104     {
105         isFading = true;
106         fadingLastCall = currentMillis;
107         totalDimmingValue = fadingInterval[0];
108         Set(TotalDimming, totalDimmingValue);
109     }
110
111 protected:
112     /// @brief Die aktuelle Gesamthelligkeit.
113     uint8_t totalDimmingValue = 0;
114
115     /// @brief Zeitpunkt des Starts des blinkens.
116     uint16_t blinkStart = 0;
117     /// @brief Dauer des blinkenden Lichts.
118     unsigned short blinkTimeout = 100;
119
120     /// @brief Letzter Zeitpunkt des dimmens.
121     uint16_t fadingLastCall = 0;
122     /// @brief Zeit zwischen dem Dimmen zweier Werte.
123     uint8_t fadingTimeout = 10;
124     /// @brief Intervall des dimmens.
125     uint8_t fadingInterval[2] = {
126             255,    //Anfang
127             0       //Ende
128     };
129 };
130
131 #endif
```

Die Befehle für alle Geräte werden in einem Array zusammengefasst.

```
1  #ifndef COMMANDLIST_H
2  #define COMMANDLIST_H
3
4  /// @brief Liste aller zu sendenden Befehle
5  DmxCommand dmxCommandList[74] = {
6          {1, 0,     MiniMovingHead14Ch::RGB,          51400},
7          {1, 0,     MiniMovingHead14Ch::Pan,          128},
8          {1, 0,     MiniMovingHead14Ch::Tilt,         128},
9
10         {0, 0,     RgbwSpotlight8Ch::RGB,            255},
11         {0, 0,     RgbwSpotlight8Ch::TotalDimming,   255},
12         {0, 400,   RgbwSpotlight8Ch::Blink,          255},
13         {0, 800,   RgbwSpotlight8Ch::Blink,          255},
14         {0, 1100,  RgbwSpotlight8Ch::Blink,          255},
15         {0, 1580,  RgbwSpotlight8Ch::Blink,          255},
16         {0, 1950,  RgbwSpotlight8Ch::Blink,          255},
17         {0, 2350,  RgbwSpotlight8Ch::Blink,          255},
18         {0, 2700,  RgbwSpotlight8Ch::Blink,          255},
19         {0, 2990,  RgbwSpotlight8Ch::BlinkTimeout,   255},
```

```
20          {0, 3000,  RgbwSpotlight8Ch::Blink,          255},
21          {0, 3400,  RgbwSpotlight8Ch::BlinkTimeout,   100},
22          {0, 3550,  RgbwSpotlight8Ch::Blink,          255},
23          {0, 3900,  RgbwSpotlight8Ch::Blink,          255},
24          {0, 4200,  RgbwSpotlight8Ch::Blink,          255},
25          {0, 4700,  RgbwSpotlight8Ch::TotalDimming,   0},
26          {0, 4700,  RgbwSpotlight8Ch::RGB,            65280},
27
28          {1, 4700,  MiniMovingHead14Ch::Speed,        251},
29          {1, 4700,  MiniMovingHead14Ch::Effect,       255},
30          {1, 4700,  MiniMovingHead14Ch::Tilt,         255},
31          {1, 4700,  MiniMovingHead14Ch::Effect,       190},
32          {1, 16000, MiniMovingHead14Ch::Speed,        0},
33          {1, 16000, MiniMovingHead14Ch::Effect,       0},
34          {1, 16000, MiniMovingHead14Ch::Effect,       255},
35          {1, 16000, MiniMovingHead14Ch::Pan,          255},
36          {1, 16000, MiniMovingHead14Ch::Tilt,         255},
37          {1, 16000, MiniMovingHead14Ch::RGB,          16711680},
38
39          {0, 16000, RgbwSpotlight8Ch::RedDimming,     255},
40          {0, 16000, RgbwSpotlight8Ch::BlueDimming,    0},
41
42          {1, 18000, MiniMovingHead14Ch::Speed,        100},
43          {1, 18000, MiniMovingHead14Ch::Pan,          0},
44          {1, 18000, MiniMovingHead14Ch::Tilt,         0},
45          {1, 18000, MiniMovingHead14Ch::GreenDimming, 100},
46
47          {1, 21000, MiniMovingHead14Ch::Pan,          200},
48          {1, 21000, MiniMovingHead14Ch::Tilt,         180},
49          {1, 21000, MiniMovingHead14Ch::RedDimming,   0},
50          {1, 21000, MiniMovingHead14Ch::WhiteDimming, 100},
51
52          {0, 22000, RgbwSpotlight8Ch::GreenDimming,   255},
53          {0, 22001, RgbwSpotlight8Ch::RedDimming,     0},
54
55          {1, 24000, MiniMovingHead14Ch::Pan,          0},
56          {1, 24000, MiniMovingHead14Ch::Tilt,         0},
57          {1, 24000, MiniMovingHead14Ch::BlueDimming,  100},
58
59          {1, 26000, MiniMovingHead14Ch::Pan,          80},
60          {1, 26000, MiniMovingHead14Ch::Tilt,         190},
61          {1, 26000, MiniMovingHead14Ch::BlueDimming,  0},
62          {1, 26000, MiniMovingHead14Ch::GreenDimming, 0},
63          {1, 26000, MiniMovingHead14Ch::RGB,          0},
64          {1, 26000, MiniMovingHead14Ch::WhiteDimming, 255},
65
66          {0, 26000, RgbwSpotlight8Ch::WhiteDimming,   200},
67
68          {1, 27000, MiniMovingHead14Ch::Pan,          40},
69          {1, 27000, MiniMovingHead14Ch::Tilt,         150},
70          {1, 27000, MiniMovingHead14Ch::WhiteDimming, 0},
71          {1, 27000, MiniMovingHead14Ch::RGB,          255},
72
73          {1, 28000, MiniMovingHead14Ch::Pan,          80},
74          {1, 28000, MiniMovingHead14Ch::Tilt,         190},
75          {1, 28000, MiniMovingHead14Ch::RGB,          16711680},
76
77          {1, 29000, MiniMovingHead14Ch::Pan,          255},
78          {1, 29000, MiniMovingHead14Ch::Tilt,         0},
79          {1, 29000, MiniMovingHead14Ch::RGB,          65280},
80
81          {0, 30000, RgbwSpotlight8Ch::TotalDimming,   255},
```

```
82
83          {1, 30500, MiniMovingHead14Ch::Speed,        190},
84          {1, 30501, MiniMovingHead14Ch::Pan,          128},
85          {1, 30501, MiniMovingHead14Ch::Tilt,         128},
86
87          {0, 31000, RgbwSpotlight8Ch::Fade,           30},
88          {1, 31000, MiniMovingHead14Ch::Fade,         30},
89          {0, 32800, RgbwSpotlight8Ch::Stop,           0},
90          {1, 32800, MiniMovingHead14Ch::Stop,         0}
91  };
92
93  #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) :
        ↪ RgbwSpotlight8Ch(address) {
10          pinMode(Address, OUTPUT);
11      }
12
13      /// @brief Überschreibt den DMX-Ausgang mit lokalen Pins.
14      /// @param channel Die Funktion oder der Kanal.
15      /// @param value Der Wert der Funktion
16      void Set(unsigned short channel, uint8_t value) override {
17          switch (channel) {
18              case 0:
19                  if (value <= 100) {
20                      digitalWrite(Address, LOW);
21                  } else {
22                      digitalWrite(Address, HIGH);
23                  }
24                  break;
25              default:
26                  break;
27          }
28      }
29
30      /// @brief Überschreibt den Reinigungsprozess um Pins benutzen zu können.
31      /// @param currentMillis Der aktuelle Zeitstempel.
32      void CleanUp(uint16_t currentMillis) override {
33          if (blinkOn && (currentMillis - blinkStart) > 100) {
34              digitalWrite(Address, LOW);
35              blinkOn = false;
36          }
37      }
38  };
39
40  #endif
```

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.

2.5 Aufgabe 4.5

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

Durchführung Mithilfe des Befehls wird der Arduino zurückgesetzt:

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

avrdude: [...]

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: safemode: lfuse reads as 0
avrdude: safemode: hfuse reads as 0
avrdude: safemode: efuse reads as 0
avrdude: reading input file "/mnt/datadisk/dmx_default.hex"
avrdude: writing flash (1482 bytes):

Writing | ################################################## | 100% 0.45s

avrdude: 1482 bytes of flash written
avrdude: verifying flash memory against /mnt/datadisk/dmx_default.hex:
avrdude: load data flash data from input file /mnt/datadisk/dmx_default.hex:
avrdude: input file /mnt/datadisk/dmx_default.hex contains 1482 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.34s

avrdude: verifying ...
avrdude: 1482 bytes of flash verified

avrdude: safemode: lfuse reads as 0
avrdude: safemode: hfuse reads as 0
avrdude: safemode: efuse reads as 0
avrdude: safemode: Fuses OK (E:00, H:00, L:00)

avrdude done.  Thank you.
```

Schlussfolgerung Der Arduino befindet sich in seinem Ausgangszustand und wurden Ordnungsgemäß zurück geräumt.