Versuch V2

C752 Digitaltechnik

Abnahme: 16. Januar 2024
Stand: 14. 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:	11 363 Bd
Anzahl Stopbits:	2
Fehler:	Nein
Daten:	

Schlussfolgerung

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

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.

```
#define DEMO 0          //Demomodus AUS bei 0
#define BPM_IN_MS 484   //Quotient aus 60.000ms und BPM

#include <Arduino.h>
#include "RgbwSpotlight8Ch.h"
#include "MiniMovingHead14Ch.h"

//Geräte aus Klassen mit DMX Adresse instanzieren
#if DEMO == 0
/// @brief Enthält alle Spotlichter.
RgbwSpotlight8Ch spotlights[2] = {
        {RgbwSpotlight8Ch(1)},
        {RgbwSpotlight8Ch(37)}
};
/// @brief Enthält alle Movingheads.
MiniMovingHead14Ch movingHeads[2] = {
        {MiniMovingHead14Ch(9)},
        {MiniMovingHead14Ch(23)}
};

//Geräte aus Klassen mit Pin-Nummern instanzieren
#else

#include "RgbwSpotlight8ChDemo.h"
#include "MiniMovingHead14ChDemo.h"

/// @brief Enthält alle Spotlichter als LED.
RgbwSpotlight8ChDemo spotlights[2] = {
        {RgbwSpotlight8ChDemo(2)},
        {RgbwSpotlight8ChDemo(4)}
};
/// @brief Enthält alle Movingheads als LED.
MiniMovingHead14ChDemo movingHeads[2] = {
        {MiniMovingHead14ChDemo(3)},
        {MiniMovingHead14ChDemo(5)}
};
#endif

/// @brief Zeitstempel des letzten erkannten Beats.
unsigned int previousMillis = 0;

/// @brief Zeitstempel des Starts der Lichtshow.
unsigned int loopStartMillis = 0;

/// @brief Zeitintervalls, in denen erkannte Beats verarbeitet werden sollen.
unsigned int beatIntervals[1][2] = {
        {4770, 30000},
```

```
48  };
49
50  /// @brief Prüft bei jedem Aufruf, ob inder bereits verstrichenen Zeit ein Beat
    ↪  auftrat.
51  /// @param currentMillis Der Zeitpunkt des Aufrufs in.
52  void BeatDetector(unsigned int currentMillis) {
53      if ((currentMillis - previousMillis) >= BPM_IN_MS) {
54          previousMillis = currentMillis;
55
56          for (auto const &interval: beatIntervals) {
57              if (interval[0] <= currentMillis && currentMillis <= interval[1]) {
58                  //Event eines Beats
59                  for (auto &spotlight: spotlights) {
60                      spotlight.StartBlink(currentMillis, 255);
61                  }
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
    ↪  Ausführen von Befehlen weiter.
91  void loop() {
92      unsigned int currentMillis = millis() - loopStartMillis;
93      BeatDetector(currentMillis);
94
95      for (auto &spotlight: spotlights) {
96          spotlight.CleanUp(currentMillis);
97          spotlight.RunTick(currentMillis);
98      }
99      for (auto &movingHead: movingHeads) {
100         movingHead.CleanUp(currentMillis);
101         movingHead.RunTick(currentMillis);
102     }
103 }
```

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.

```
#ifndef DmxCommand_h
#define DmxCommand_h

/// @brief Beschreibt einen Befehl, der über DMX ausgeführt werden soll.
struct DmxCommand {
    /// @brief Der Zeitpunkt der geplanten Ausführung.
    unsigned int executionTime;
    /// @brief Der Kanal oder die Funktion, welche bedient wird.
    int function;
    /// @brief Der Wert, welcher die Funktion verändert.
    unsigned char value;
};
#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.

```
#ifndef DmxDevice_h
#define DmxDevice_h

#include "DmxSimple.h"
#include "DmxCommand.h"
#include <Arduino.h>

/// @brief Basisklasse für alle DMX-Geräte
class DmxDevice {
public:
    explicit DmxDevice(unsigned short address) : Address(address) {}

    /// @brief Die Adresse des Geräts
    unsigned short Address;

    /// @brief Erzeugt ein DMX-Signal an einem Kanal des Geräts.
    /// @param channel Der Kanal des Geräts, beginnend bei Kanal 0.
    /// @param value Der Wert des Signals im Bereich von 0 bis 255.
    virtual void Set(int channel, unsigned char value) {
        DmxSimpleClass::write(static_cast<int>(Address + channel), value);
    };

    /// @brief Verarbeitet Befehle zu ihren Zeitpunkten.
    /// @param currentMillis Der aktuelle Zeitstempel.
    virtual void RunTick(unsigned int currentMillis) = 0;

    /// @brief Bereinigt Spuren vergangener Befehle oder führt diese fort, sofern
    ↪  sie mehrere Aufrufe benötigen.
    /// @param currentMillis Der aktuelle Zeitstempel.
    virtual void CleanUp(unsigned int currentMillis) = 0;

    /// @brief Werzeug für das debuggen von Befehlen.
    /// @param currentMillis Der aktuelle Zeitstempel.
    static void PrintTimeToSerial(unsigned int currentMillis) {
        Serial.print("Time elapsed: ");
        Serial.print(currentMillis);
        Serial.println("ms");
    }
};

#endif
```

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

```
#ifndef RgbwSpotlight8Ch_h
#define RgbwSpotlight8Ch_h

#include "DmxDevice.h"
#include "DmxCommand.h"

/// @brief Beschreibt einen DMX-Scheinwerfer mit acht Kanälen.
class RgbwSpotlight8Ch : public DmxDevice {
public:
    explicit RgbwSpotlight8Ch(unsigned short address) : DmxDevice(address) {}

    /// @brief Alle Funktionen, welche das Gerät unterstützt.
    enum Functions {
        TotalDimming,
        RedDimming,
        GreenDimming,
        BlueDimming,
        WhiteDimming,
        TotalStrobe,
        FuncSelection,
        FuncSpeed,

        Blink = 100,
        Fade = 101,
        BlinkTimeout = 200,

        Stop = 300,
        PrintTime = 301
    };
    bool blinkOn = false;
    bool isFading = false;

    /// @brief List der Befehle des Geräts.
    DmxCommand commandList[25] = {
            {100,   BlueDimming,  255},
            {200,   TotalDimming, 100},
            {400,   Blink,        255},
            {800,   Blink,        255},
            {1100,  Blink,        255},
            {1580,  Blink,        255},
            {1950,  Blink,        255},
            {2350,  Blink,        255},
            {2700,  Blink,        255},
            {2990,  BlinkTimeout, 255},
            {3000,  Blink,        255},
            {3400,  BlinkTimeout, 100},
            {3550,  Blink,        255},
            {3900,  Blink,        255},
            {4200,  Blink,        255},
            {4700,  TotalDimming, 0},
            {4701,  RedDimming,   100},
            {16000, RedDimming,   255},
            {16001, BlueDimming,  0},
            {22000, GreenDimming, 255},
            {22001, RedDimming,   0},
            {26000, WhiteDimming, 200},
            {30000, TotalDimming, 255},
            {31000, Fade,         30},
            {32800, Stop,         0}
```

```
60      };
61
62      /*DmxCommand commandList[3] = {
63              {100,   BlueDimming,  255},
64              {100,   TotalDimming,  255},
65              {100,   Fade,  10}
66      };*/
67
68      void RunTick(unsigned int currentMillis) override {
69          DmxCommand cmd = commandList[commandIndex];
70          if (cmd.executionTime < currentMillis) {
71              if (cmd.function != Stop) {
72                  if (cmd.function < 200) {
73                      cmd.value = static_cast<unsigned char>(cmd.value);
74                  }
75
76                  switch (cmd.function) {
77                      case PrintTime:
78                          PrintTimeToSerial(currentMillis);
79                      case Blink:
80                          StartBlink(currentMillis, cmd.value);
81                          break;
82                      case Fade:
83                          fadingTimeout = cmd.value;
84                          StartFading(currentMillis);
85                      case BlinkTimeout:
86                          blinkTimeout = cmd.value;
87                      case TotalDimming:
88                          totalDimmingValue = cmd.value;
89                          Set(TotalDimming, totalDimmingValue);
90                      default:
91                          Set(static_cast<Functions>(cmd.function), cmd.value);
92                          break;
93                  }
94                  commandIndex++;
95              } else {
96                  Set(TotalDimming, 0);
97              }
98          }
99      };
100
101     void CleanUp(unsigned int currentMillis) override {
102         // Blinken beenden
103         if (blinkOn && (currentMillis - blinkStart) > blinkTimeout) {
104             Set(TotalDimming, totalDimmingValue);
105             blinkOn = false;
106         }
107
108         // Dimmen fortsetzen falls aktiv
109         if (isFading && (currentMillis - fadingLastCall) > fadingTimeout) {
110             totalDimmingValue--;
111             fadingLastCall = currentMillis;
112             Set(TotalDimming, totalDimmingValue);
113             isFading = fadingInterval[1] < totalDimmingValue;
114         }
115     }
116
117     /// @brief Bereitet das Blinken des Lichts vor.
118     /// @param currentMillis Der aktuelle Zeitstempel.
119     /// @param value Die Stärke des blinkenden Lichts.
120     void StartBlink(unsigned int currentMillis, unsigned char value) {
121         blinkOn = true;
```

```
122             blinkStart = currentMillis;
123             Set(TotalDimming, value);
124         }
125
126         /// @brief Bereitet das dimmen des Lichts vor.
127         /// @param currentMillis Der aktuelle Zeitstempel.
128         void StartFading(unsigned int currentMillis) {
129             isFading = true;
130             fadingLastCall = currentMillis;
131             totalDimmingValue = fadingInterval[0];
132             Set(TotalDimming, totalDimmingValue);
133         }
134
135     protected:
136         /// @brief Die aktuelle Zeile in der Befehlsliste.
137         unsigned short commandIndex = 0;
138         /// @brief Die aktuelle Gesamthelligkeit.
139         unsigned char totalDimmingValue = 0;
140
141         /// @brief Zeitpunkt des Starts des blinkens.
142         unsigned int blinkStart = 0;
143         /// @brief Dauer des blinkenden Lichts.
144         unsigned short blinkTimeout = 100;
145
146         /// @brief Letzter Zeitpunkt des dimmens.
147         unsigned int fadingLastCall;
148         /// @brief Zeit zwichen dem Dimmen zweier Werte.
149         unsigned char fadingTimeout = 10;
150         /// @brief Intervall des dimmens.
151         unsigned char fadingInterval[2] = {
152                 255,    //Anfang
153                 0       //Ende
154         };
155     };
156
157     #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(int channel, unsigned char 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(unsigned int 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.