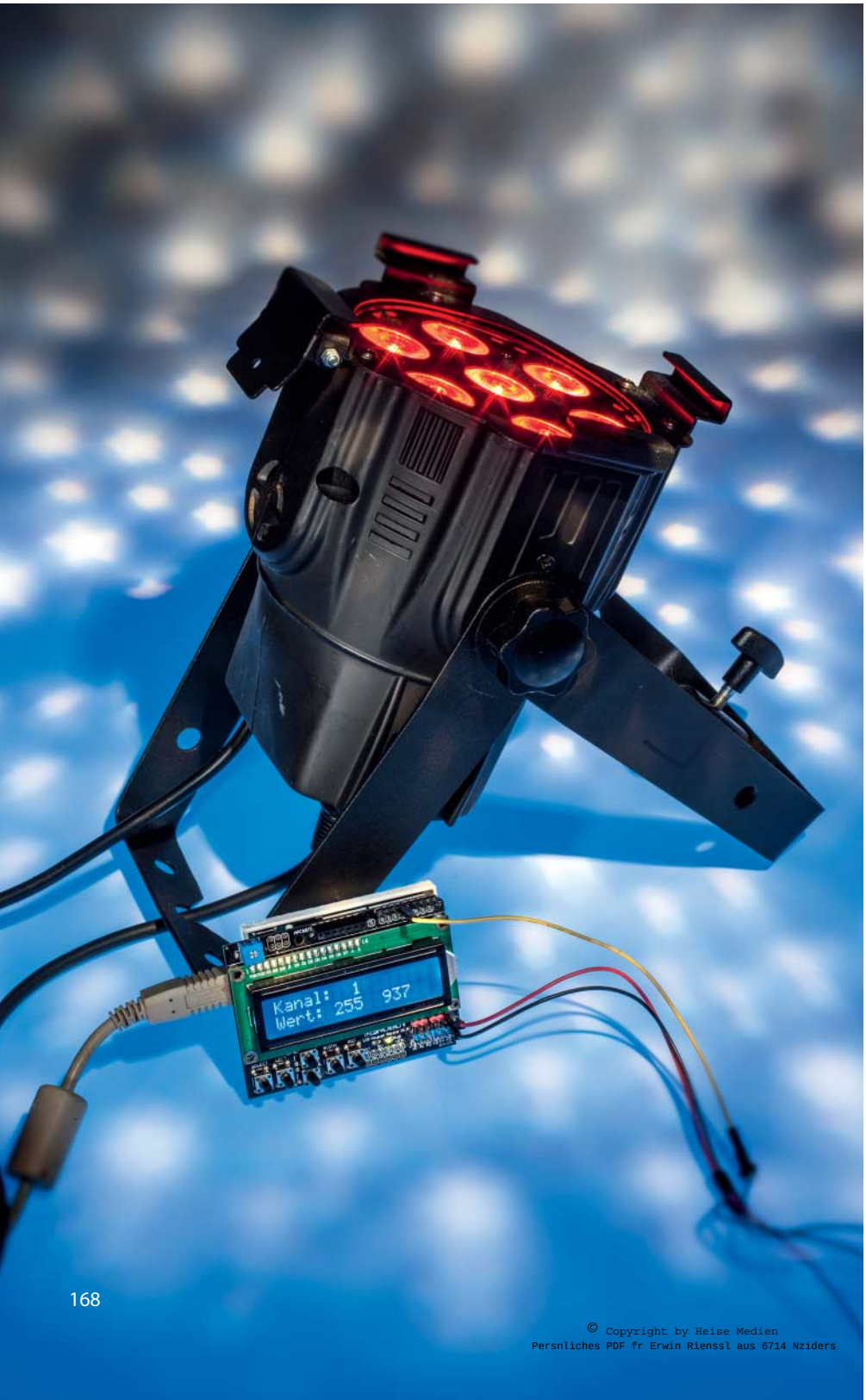
Pascal Jacob, Tim Gerber

Licht mit X

DMX-Lichtsteuerpult mit Arduino selbst bauen

Ob für Hobby-Band, Freizeit-Bühne oder Partykeller: Auch bezahlbare Scheinwerfer und Effektgeräte lassen sich fernsteuern. Billigen DMX-Pulten fehlen jedoch komfortable Funktionen, die sich per Arduino leicht selbst realisieren lassen.

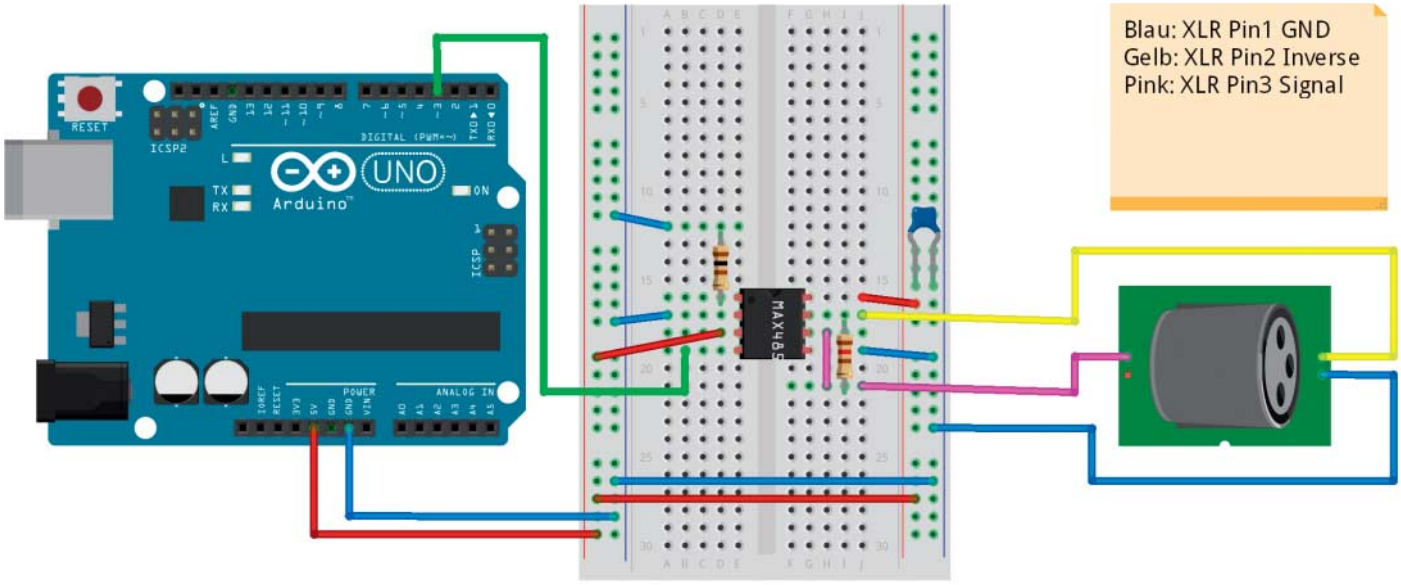


Wenn auf der Bühne oder im Partykeller eine fetzige Lichtshow zu sehen sein soll, braucht man am Steuerpult etwas mehr als nur sechs oder zwölf Schieberegler, wie sie einfache DMX-Lichtsteuerpulte ab 40 Euro aufweisen. Für diesen Preis kann man bereits den Kern eines selbst gebauten, leistungsfähigen DMX-Pultes auf Arduino-Basis konstruieren, das sich nahezu beliebig erweitern und auf die eigenen Bedürfnisse anpassen lässt. Solcher Komfort kostet bei fertigen Produkten schnell mehrere Hundert Euro.

Das DMX-Protokoll erfreut sich in der Veranstaltungstechnik seit mehreren Jahrzehnten trotz einiger Nachteile einer sehr weiten Verbreitung. DMX steht für Digital Multiplexing, das gängige Protokoll heißt DMX512. Damit können bis zu 512 Endgeräte mit jeweils einem Byte Daten über eine Datenleitung angesteuert werden. Eine Adressierung gibt es im Datenstrom nicht: Das Gerät mit der Adresse x wertet einfach das x-te Byte im Datenstrom aus. Eine günstige Möglichkeit, DMX-Geräte anzusteuern, besteht darin, sie über einen DMX-USB-Adapter mit einem PC zu verbinden. Das nötige Interface gibt es ab zirka 50 Euro und in Kombination mit einem der inzwischen zahlreichen Freeware-Programme wie Freestyler, PC-Dimmer oder DMX-Control gibt das bereits eine verwendbare Steuereinheit ab, die einen weitaus größeren Funktionsumfang bietet, als eigenständige DMX-Pulte der Preiskategorie bis 100 Euro. Nachteil dieser Lösung: Es wird immer ein kompletter PC oder Laptop benötigt und die reine Softwarelösung lässt im Vergleich zu echten Fadern bei Bedienkomfort und -geschwindigkeit doch schnell zu wünschen übrig, weil man auf PC-Tastatur, Maus und Bildschirm beschränkt ist. Ein selbst gebautes Pult lässt sich dagegen leicht mit Schiebe- oder Drehreglern ausstatten.

Technisch handelt es sich bei dem DMX-Signal um eine serielle, asynchrone Datenübertragung. Asynchron deshalb, weil Sender und Empfänger nicht über eine eigene Taktleitung synchronisiert werden, sondern über eine feste Baudrate (bei DMX 250 kBit/s) und den Rahmen aus Start- und Stoppbits. Elektrisch gesehen entspricht DMX dem Industriestandard RS-485. Dabei werden die logischen Zustände (`HIGH` und `LOW`) der Datenleitung nicht durch eine bestimmte Spannung gegen Masse repräsentiert, sondern durch die Spannungsdifferenz der beiden Datenleitungen gegeneinander. Beim RS-485-Signal ist das +2,5 Volt oder höher für `HIGH` und -2,5 Volt für `LOW`. Diese symmetrische elektrische Signalübertragung hat den Vorteil, dass sich Störsignale auf die Spannungsdifferenz kaum auswirken.

Das erforderliche serielle DMX-Signal ist mit einem Mikrocontroller, wie er auf den Arduino-Boards verbaut ist, recht leicht zu erzeugen. Da es auf die Differenz und nicht die absolute Spannung ankommt, reichen die 5 oder auch 3,3 Volt der Arduinos aus. Um aus den logischen 5-Volt-Pegeln (TTL) des Arduino ein RS-485-Signal zu erzeugen, gibt es für wenige Euro im Elektronik-Fach-



Die Schaltung für ein DMX-Interface am Arduino lässt sich mit ein paar Handgriffen auf dem Steckbrett aufbauen. Sie auf eine Lochrasterplatine zu löten stellt ebenfalls keine große Hürde dar.

handel fertige ICs wie den MAX485. Die Details der DMX-Erzeugung wie das exakte Timing und seine Besonderheiten wie das überlange Startbyte übernimmt eine fertige Bibliothek, darum muss sich der Arduino-Entwickler keine Gedanken machen. Die Schaltung für den DMX-Ausgang am Arduino ist denkbar einfach (siehe Abbildung) und leicht auf einem Steckbrett oder noch besser auf einer Lochrasterplatine zu realisieren.

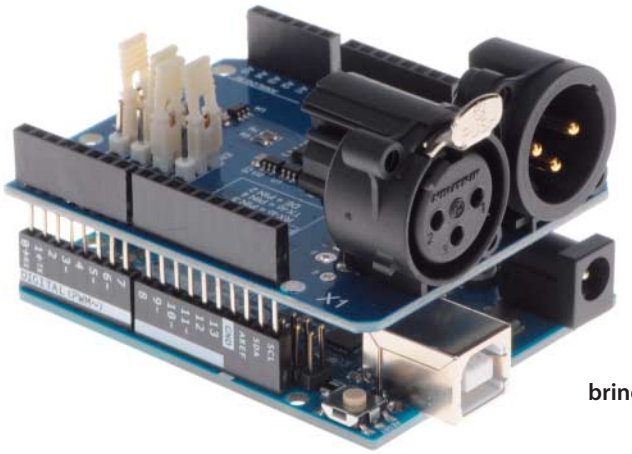
Alternativ gibt es fertige Arduino-Shields, die neben der Schaltung auch gleich die XLR-Buchsen für die DMX-Verkabelung mitbringen. Man bekommt sie in einfacher Ausführung ab etwa 20 Euro. Um 10 Euro teurere Versionen bringen Optokoppler und Spannungswandler zur galvanischen Trennung der DMX-Schaltung von der Mikrocontrollerelektronik mit und bieten dadurch mehr Ausfallsicherheit. Anders als die einfache MAX485-Schaltung kann ein DMX-Shield auch DMX-Daten empfangen.

Für ein Steuerpult, also einen DMX-Master, reichen die Sendefähigkeiten der einfachen Schaltung ohne Shield völlig aus. Um ihre Funktion zu testen, benutzt man am besten die Bibliothek DmxSimple, denn die erfüllt genau die Erwartungen, die ihr Name weckt. Über den Menüpunkt Sketch/Include Library der Arduino IDE kann man das heruntergeladene ZIP-Archiv (siehe c't-Link) in das Bibliotheksverzeichnis der Arduino-IDE entpacken. Die jüngste Version 1.6.5 der IDE benötigt danach keinen Neustart mehr. Die DmxSimple-Bibliothek ist etwas älter, deshalb muss man nach dem Import die Zeile `#include „wiring.h“` in der Datei dmxsimple.cpp mit einem Texteditor durch die Zeile `#include „arduino.h“` ersetzen. Danach kann man über Datei/Beispiele das zur DmxSimple-Bibliothek gehörende Programm FadeUp öffnen und auf den Arduino übertragen. Es sieht in etwa so aus:

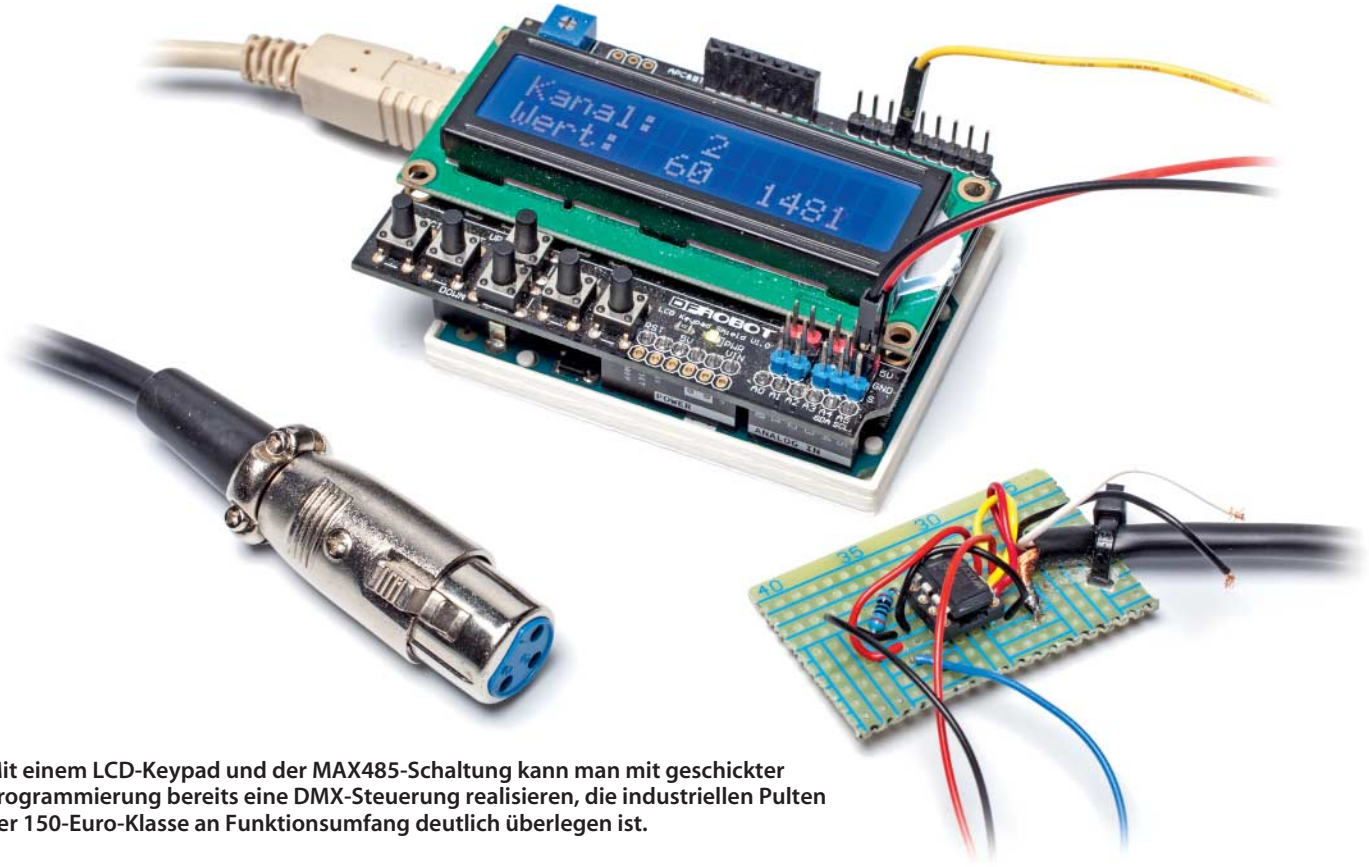
```
#include <DmxSimple.h>
#define arduinoPin 3
#define dmxChannel 1
void setup() {
DmxSimple.usePin(arduinoPin);
DmxSimple.maxChannel(1);
}
void loop() {
  for (int value = 0; value < 255; value++) {
    DmxSimple.write(dmxChannel, value);
    delay(200);
  }
delay(3000);
}
```

Man kann an diesem Beispielprogramm sehr gut die Arbeitsweise der Bibliothek studieren: Sie bietet keine Funktionen für das eigentliche Senden der DMX-Daten, sondern erledigt dies selbstständig im Hintergrund, ohne dass man sich als Programmierer darum kümmern müsste. Die Änderung der Werte erfolgt dadurch, dass man sie über entsprechende Funktionen der Bibliothek in einen von ihr reservierten Speicherbereich schreibt. Um die knappen Ressourcen einfacher Arduino-Boards wie dem Uno oder Leonardo nicht unnötig zu blockieren, empfiehlt es sich deshalb, zu Beginn des Programms die Zahl der verwendeten DMX-Kanäle mit dem Befehl `DmxSimple.maxChannel(1);` auf die tatsächlich benötigte Zahl zu begrenzen. Denn das DMX-Protokoll verlangt nicht, dass stets alle maximal möglichen 512 Bytes gesendet werden. Die Übertragung kann jederzeit abbrechen und durch erneutes Senden des überlangen Startbytes aufs Neue beginnen.

Natürlich ist das kleine Beispielprogramm alles andere als abendfüllend. Mit der Schaltung kann man aber bereits nach Lust und Laune an Stelle der stupiden Aufblenderei auch ausgefeilte Effekte über sämtliche 512 Kanäle programmieren, indem man sie munter in die Loop-Funktion hineinschreibt. Ein-



Fertige DMX-Shields für den Arduino gibt es für zirka 25 Euro. Sie können einfach auf den Arduino aufgesteckt werden und bringen die notwendigen XLR-Steckverbinder gleich mit.



Mit einem LCD-Keypad und der MAX485-Schaltung kann man mit geschickter Programmierung bereits eine DMX-Steuerung realisieren, die industriellen Pulten der 150-Euro-Klasse an Funktionsumfang deutlich überlegen ist.

griffsmöglichkeiten von außen, also durch Benutzereingaben, kann man mit dieser Schaltung aber nicht realisieren. Dafür fehlen ihr Elemente für Benutzerein- und -ausgaben wie Taster oder ein Display.

Eine einfache Variante der Benutzersteuerung besteht beispielsweise darin, den Wert eines Kanals über einen Schiebewiderstand (am besten 10 Kiloohm linear) einzustellen, der am Analog-Eingang A0 angeschlossen ist. Dazu muss man nur den Wert mit `analogRead(A0)` auslesen. Um die vom A/D-Wandler des Arduino gelieferten 10-Bit-Werte (0 bis 1023) in die fürs DMX-Protokoll benötigten 8-Bit-Werte umzuwandeln, teilt man sie einfach durch 4. Die Programmschleife sieht nun so aus:

```
void loop() {
    int value = analogRead(A0)/4;
    DmxSimple.write(dmxChannel, value);
    delay(200);
}
```

An einfache Arduino-Boards wie Uno und Leonardo kann man bis zu sechs Potis anschließen, an größere Boards wie den Mega 2560 auch zwölf. Letztlich ist man damit aber nicht wesentlich weiter als mit einem fertigen DMX-Pult für 40 Euro, das zusätzlich zu den sechs Kanalreglern immerhin noch einen Master-Regler mitbringt und bereits in einem fertigen Gehäuse sitzt. Zum Glück geht mit einem Arduino aber auch noch deutlich mehr, denn die Schaltung lässt sich nahezu unbegrenzt um diverse Eingabemöglichkeiten und andere Peripherie wie Display, Touchscreen, SD-Kartenslot und dergleichen mehr erweitern. Und dann spielt die Selbstbau-Lösung gegenüber Angeboten wie einem DMX-Interface mit einfacher Steuersoftware ihre Stärken aus.

Abendmenü

Dabei ist es vorteilhaft, auf analoge Eingaben über Potis eher zu verzichten, außer vielleicht für einen Master-Regler, und stattdessen die Werte gleich digital, beispielsweise über Taster einzugeben. Dafür kann man ein fertiges LCD-Keypad für den Arduino benutzen, eine Kombination aus Display und ein paar Tastern für eine Menü-Steuerung. Damit kommt man recht schnell und einfach zu einem kleinen DMX-Pult mit eleganter Digitaleingabe und einem größeren Funktionsumfang. Wir haben uns dazu ein LCD-Keypad vom chinesischen Anbieter DFRobot mit einem zweizeiligen Display und fünf Menütastern besorgt, das im Handel für knapp 13 Euro zu bekommen ist. Vergleichbare Shields gibt es von diversen Anbietern. Für das Display belegt das Shield die Digitalpins 4 bis 10. Die fünf Taster sind in einer geschickten Schaltung über verschiedene Widerstände an einen einzigen Analogpin angeschlossen. Anhand der verschiedenen resultierenden Spannungen an dem Pin kann das Programm den jeweils gedrückten Taster ermitteln und darauf reagieren.

Der Hersteller liefert für das LCD-Keypad ein Beispielprogramm (siehe c't-Link), das bereits das Gerüst für ein komplettes Menü nebst Display-Ausgabe bereitstellt. Das haben wir so ausgefüllt und angepasst, dass die Links- und Rechts-Taste die Auswahl eines von insgesamt zwölf Scheinwerferkanälen erlaubt, deren Helligkeit dann jeweils über die Auf- und Abwärts-Taste geändert werden kann. Das Display zeigt den angewählten Kanal und den zugehörigen Helligkeitswert an.

Dieser Aufbau ließe sich ohne weitere Hardware um Funktionen erweitern, indem man einfach das Programm weiterentwickelt und die Menüstruktur verfeinert. Allerdings ist das nicht sonderlich ergonomisch, wenn eine größere Anzahl von Kanälen gesteuert werden soll und darüber nicht nur einfache Scheinwerfer, sondern auch RGB-LEDs oder gar Moving Heads, also ferngesteuert bewegbare Scheinwerfer angesteuert werden.

Um solche LED-Effektscheinwerfer über DMX anzusteuern, braucht man mehrere DMX-Adressen, die man am Pult zusammenfassen und gemeinsam bedienen will. Günstige Moving Heads benötigen nur je eine Adresse für die horizontale (Pan) und vertikale (Tilt) Bewegung sowie die Helligkeiten. Eine Bewegung über den Pan- oder Tilt-Kanal kann beispielsweise mit der oben dargestellten For-Schleife realisiert werden. Für eine Kreisbewegung ist zuvor noch die Berechnung der X- und Y-Koordinate, also der Pan- und Tiltwerte auf dem Kreisbogen notwendig. Um ein Moving Head kreisen zu lassen, eignen sich die Rechts- und Links-Taster am LCD-Keypad:

```
case btnRIGHT {
    if(position >= numberOfCirclePoints)
    position =0;
    else
    position++;
```

```
    int xPosition = cos(position * 2 * PI /
numberOfCirclePoints) * radius + xCenter;
    int yPosition = sin(position * 2 * PI /
numberOfCirclePoints) * radius + yCenter;
DmxSimple.write(panChannel, xPosition);
DmxSimple.write(tiltChannel, yPosition);
}
```

Für Moving Heads oder eine größere Zahl normaler Scheinwerfer bietet das zweizeilige Display unseres Minipults nicht genug Platz. Und statt über die etwas mickrigen Taster des Keypads möchte man bestimmte Werte und Funktionen lieber über digitale Drehencoder eingeben. Letztere sehen ganz ähnlich aus wie ein Poti und werden auch Drehimpulsgeber genannt. Sie geben bei Drehbewegungen an ihrem Schaft über zwei Kontakte Schaltimpulse, aus deren Abfolge ein Mikrocontrollerprogramm Drehrichtung und -winkel ablesen kann. Meist haben sie noch einen dritten Kontakt, der durch Drücken auf den Schaft wie ein Taster betätigt wird. Auf der Arduino-Homepage und in einschlägigen Foren gibt es ausführliche Anleitungen, wie man solche Dreh-Encoder oder auch einen schicken, farbigen Touchscreen an einen Arduino anschließt (siehe c't-Link) und über die zugehörigen Bibliotheken in eigene Programme einbindet.

Großprojekt

Für ein leistungsfähiges Lichtpult für 512 DMX-Kanäle, Presets und automatischen Überblendungen reichen die Fähigkeiten eines einfachen Arduino wie dem Uno oder Leonardo nicht aus. Für 20 Euro mehr erhält man mit dem Arduino Mega2560 einen deutlich leistungsfähigeren Controller. Um eine größere Zahl Bedienelemente anzuschließen, kann man ihn mit anderen Controllern wie dem ATMega 328P (Arduino Uno oder Nano) erweitern. Wie das gehen kann, zeigt das Beispiel eines an der Hochschule für Technik, Wirtschaft und Gestaltung (HTWG) in Konstanz im Rahmen einer Bachelor-Arbeit entwickelte und gebaute „LightOn1“. Es besteht aus 98 Tastern kombiniert mit acht Fadern, einem 7-Zoll-Farbtouchscreen und drei Drehimpulsgebern in einem 19-Zoll-Gehäuse. Im Inneren beherbergt das Pult insgesamt acht Baugruppen. Für die Auswertung der Eingaben und Rückmeldung über die in den Tastern eingebauten LEDs kommen vier Arduino Nano respektive ATmega 328P-Controller und ein Arduino Mega 2560R3 (ATmega 2560) zum Einsatz. Der Mega bildet das Herzstück, behandelt Ein- und Ausgaben des Touchscreens und erzeugt letztlich auch das DMX-Signal.

Sobald einer der Arduino Nanos eine Interaktion erkannt hat, sendet er einen Datensatz mit Informationen über die Aktion mittels serieller Schnittstelle (TTL-UART) an den Arduino Mega. Diese Datenübertragung bietet gegenüber einem Bussystem wie I²C den Vorteil, dass es sich um mehrere unabhängige Punkt-zu-Punkt Verbindungen handelt, was die Fehlersuche deutlich erleichtert. Da

Im DMX-Pult „LightOn1“ der HTWG Konstanz stecken fünf Arduinos, 98 Taster, acht Fader und ein 7-Zoll-Touchscreen.

die Arduinos mit einem UART-Empfangspuffer ausgestattet sind, ist es zudem möglich, Daten zu übertragen, ohne dass der Empfänger diese wie bei I²C anfragen oder auf das vollständige Eintreffen der Daten warten muss. Der Entwickler muss sich also nicht mit dem Timing der Nachrichten und erst recht nicht mit dem ziemlich komplexen Kapitel Busmanagement auseinandersetzen.

Auf diese Weise ist es dem Arduino Mega 2560 also möglich, auf seinen vier seriellen Schnittstellen quasi-parallel zur laufenden Schleife des Hauptprogrammes die Daten zu empfangen und in die zur jeweiligen Schnittstelle gehörenden Ringpuffer zu schreiben, ohne dass man sich darum in seinem Hauptprogramm kümmern müsste. Der Mega schreibt alle an einer Schnittstelle empfangenen Daten nacheinander in den Puffer. Sollte die Standardpuffergröße von 64 Bytes überschritten werden, wird das erste Byte des Speichers überschrieben. Dies stellt sicher, dass alle Daten in den Puffer kommen, bringt jedoch zwei Probleme mit sich: Nachrichten, die nicht sofort nach dem Empfang verarbeitet werden, dürfen maximal 64 Byte groß sein. Außerdem ist es nicht möglich, auf Anhieb den Anfang einer übertragenen Nachricht im Ringpuffer zu finden.

Um das erste Problem zu lösen, wäre es möglich, die Puffergröße zu erhöhen. Die bessere Lösung ist es, seine Nachrichten parallel zum Empfang auch zu verarbeiten oder die Nachrichtengröße auf 64 Byte zu beschränken, etwa indem man mehrere Tastenzustände bitweise in Ganzzahlvariablen speichert und überträgt.

Um den Beginn der Nachricht im Ringpuffer zu erkennen, muss die Nachricht mit einem Anfang (Header), Schluss (Trailer) und im Idealfall auch mit einer Prüfsumme zur Fehlererkennung ausgestattet werden. Das überlässt man der gut dokumentierten Bibliothek EasyTransfer von Bill Porter (siehe c't-Link). Diese sendet alle in einer Datenstruktur (struct) enthaltenen Daten wahlweise über eine native serielle Schnittstelle, eine per Software emulierte oder über den I²C-Bus und kümmert sich auf Sende- wie Empfangsseite um Header, Trailer und Prüfsumme. Somit ist eine Kommunikation zwischen zwei Arduinos bereits mit wenigen Codezeilen implementierbar, ohne dass die Programmausführung durch ständiges Lesen des Empfangspuffers blockiert wäre.

Dazu muss man auf dem Sender- und Empfänger-Arduino eine identische Datenstruktur verwenden. Im Hauptprogramm des zentralen Arduino legt man mit `ET.begin(details(mydata), &Serial);` ein Easy-Transfer-Objekt an. Im Hauptprogramm genügt dann der Aufruf `if(ET.receiveData()) for (int x = 0; x < 6; x++) DmxSimple.Write(x+12, myData.Channels[x]);` um zum Beispiel die Werte von sechs am Analogport des sendenden Arduino angeschlossenen Fadern zu empfangen und als Werte über die DMX-Kanäle 13 bis 18 auszugeben. So können die Arduinos untereinander nahezu beliebig Informationen austauschen, ohne dass das Hauptprogramm ständig den Zustand bestimmter Eingänge abfragen müsste oder die Eingaben sonst verloren gehen. Diese Technik, mehrere Arduinos miteinander zu verbinden, eignet sich deshalb bestens für die Erweiterung eines selbst gebauten und erweiterbaren Lichtsteuerpultes – aber auch für andere Projekte, die die Fähigkeiten eines einzelnen Arduino überfordern würden.

(tig@ct.de)

Literatur

[1] Ulrich Hilgefort, Funklicht, DMX-Gerätschaften mit preiswerter 2,4-GHz-Funktechnik steuern, c't 21/15, S. 152

[2] Max Keller, Faszination Licht, Prestel Verlag, ISBN: 978-3-7913-4372-3, 85 Euro

[3] Roland Greule, Licht und Beleuchtung im Medienbereich, Carl Hanser Verlag, ISBN: 978-3-446-43479-0, 30 Euro

c't Beispielcode und Weiterführendes: ct.de/y43b