Arduino verstehen und anwenden

Teil 21: Audiotechnik und Sound-Wiedergabe

Im letzten Artikel zu dieser Serie wurden dem Arduino verschiedene Töne und Klänge entlockt. Bis hin zu einer synthetischen Spracherzeugung reichen die Möglichkeiten, die bereits ein einfacher AVR-Mikrocontroller auf dem Arduino bietet.

Dabei wurden die Audioausgaben jedoch immer direkt vom Prozessor erzeugt. In diesem Artikel soll es dagegen um die Wiedergabe von aufgezeichneten Tönen gehen. In einem ersten Schritt wird dazu gezeigt, wie man den Arduino als Mini-Soundkarte einsetzen kann. Dazu werden Daten über die serielle Schnittstelle an den Arduino gesendet. Dort werden sie in analoge Signale umgewandelt. Diese können dann über einen Lautsprecher wiedergegeben werden. Natürlich darf man bei dieser einfachen Anwendung keine Hi-Fi-Ergebnisse erwarten. Für eine einfache Ausgabe von WAV-Dateien ist die Methode aber durchaus brauchbar.

In einem zweiten Schritt sollen dann sogar MP3-Dateien abgespielt werden. Dazu reicht die Rechenleistung des Arduinos selbst nicht mehr aus. Deshalb wird hierfür ein Shield verwendet. Dieses verfügt über einen speziellen Decoder, der die im MP3-Format codierten Daten in analoge Tonsignale umwandelt. Zusätzlich enthält das Shield auch noch einen Audioverstärker, sodass man direkt Kopfhörer oder sogar kleine Lautsprecher anschließen kann. Damit lässt sich eine hervorragende Klangqualität erzielen, die für die meisten praktischen Anwendungen ausreichend sein sollte.

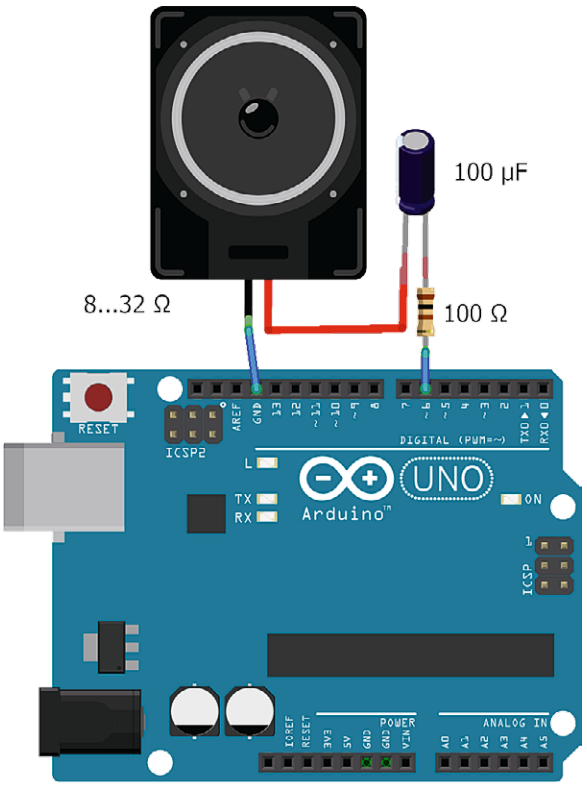

Bild 1: Einfache Audioausgabe mit dem Arduino

Der Arduino als Mini-Soundkarte

Im ersten Projekt wird gezeigt, wie mit geringem Hardware-Aufwand eine auf einem PC oder Laptop gespeicherte WAV-Datei mit einem Arduino abgespielt werden kann. Im einfachsten Fall benötigt man hierfür neben dem Arduino nur drei weitere Bauteile:

- 1x 100 µF-Kondensator
- 1x Widerstand oder Trimmer (ca. 100 Ω)
- 1x Lautsprecher (8–32 Ω)

Bild 1 zeigt den zugehörigen Schaltplan. Die gesamte Schaltung kann leicht auf einem kleinen Breadboard aufgebaut werden.

Um mit einer minimalen Anzahl von Bauelementen auszukommen, wurde auf einen Tiefpass am PWM-Ausgang des Arduinos verzichtet.

Möchte man die Tonqualität verbessern, empfiehlt es sich ohnehin, einen kleinen Audioverstärker einzusetzen. Ein passender Schaltungsvorschlag findet sich im letzten Artikel dieser Serie (Teil 20: Digitale Soundeffekte und Synthesizer, ELV Journal Feb./März).

Hier wird stattdessen ein kleiner Lautsprecher direkt über einen Elko und einen Vorwiderstand angeschlossen. Zu beachten ist, dass an diesen einfachen Audioausgang kein kommerzieller Verstärker und keine Aktivboxen angeschlossen werden dürfen. Die im Signal vorhandenen hochfrequenten Störungen aus der Pulsweitenmodulation des Arduinos könnten diese Audiogeräte beschädigen.

WAV-Dateien konvertieren

Um die WAV-Dateien auf dem Arduino abspielen zu können, müssen diese ein spezielles Format aufweisen. Die Daten sollten im RIFF-WAVE-Format/PCM mit den folgenden Parametern vorliegen:

- 8-bit
- mono
- 11,025 kHz Samplingrate

Wenn die abzuspielende Datei noch nicht in diesem Format vorliegt, kann sie beispielsweise mit Audacity konvertiert werden. Dieses äußerst nützliche Programm kann unter [1] kostenlos aus dem Internet geladen werden.

Die Datei wird in Audacity geöffnet und unten links unter *Projektrate* dann 11.025 Hz ausgewählt.

Um bei Stereoformat auf Mono zu wechseln, klickt man im Kästchen mit dem Dateinamen auf den kleinen Pfeil und wählt *Stereotonspur aufteilen*.

Anschließend kann man z. B. den rechten Tonkanal durch Klick auf das *X* löschen und beim linken Tonkanal durch erneuten Klick auf den kleinen Pfeil *Mono* auswählen. Das Speichern der Datei erfolgt

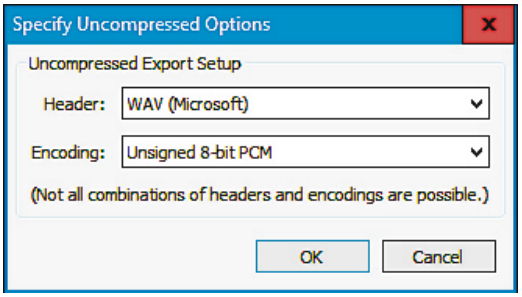

Bild 2: Exportieren der WAV-Datei

unter *Export Audio*. Die Datei muss im unkomprimierten Exportformat

- WAV
- 8-bit PCM

gespeichert werden (siehe Bild 2).
Die Datei sollte jetzt im korrekten Format vorliegen.

Ein Sketch verwandelt den Arduino in eine Soundkarte

Die WAV-Dateien müssen binär als kontinuierlicher Datenstrom, also ohne Pausen zwischen den Datenbytes, an den Arduino übertragen werden. Da beim Arduino ein virtueller COM-Port via USB geöffnet wird, ist es wichtig, dass das Sendetool die Daten nicht in einzelnen Bytes, sondern blockweise an den COM-Port-Softwaretreiber übergibt, da es sonst zu Lücken in der Wiedergabe kommen kann.

Hierfür kann z. B. das Freewaretool HTerm [2] verwendet werden. Zunächst wird in HTerm in der Auswahlbox *Port* der entsprechende COM-Port des Arduinos ausgewählt. Die serielle Schnittstelle muss dann auf

- 115.200 Baud,
- 8 Datenbits,
- 1 Stoppbit,
- keine Parität,
- keine Flusskontrolle

eingestellt werden (siehe Bild 3).

Durch einen Klick auf den Button *Connect* wird die Schnittstelle geöffnet.

Nun kann durch Klick auf den Button *Send file* die gewünschte WAV-Datei ausgewählt und dann durch Aktivieren von *Start* übertragen werden (Bild 4). Aus dem Lautsprecher sollte nun die WAV-Datei zu hören sein. Je nach Auslastung des PCs kann es z. B. beim Verschieben von Fenstern zu Lücken im Datenstrom und damit zu hörbaren Aussetzern kommen. Der Sketch dazu sieht so aus:

```
// wavPlayer.ino

void setup()
{ Serial.begin(115200);
  pinMode(6, OUTPUT);
  TCCR0A=0x83; TCCR0B=0x01;
  TCNT0=0x00;
  OCR0A=0x00; OCR0B=0x00;
}

void loop()
{ for(int i = 0; i < 44;  i++)
  {  // skip 44 bytes header
     while(Serial.available() == 0);
     Serial.read();
  }

  while(1)
  { while(Serial.available() == 0);
     OCR0A = (unsigned char)Serial.read();
   }
}
```

Das zu diesem Artikel verfügbare Download-Paket (siehe „Download-Paket zum Artikel" am Ende des Beitrags) enthält zwei Probedateien im korrekten Format. Damit kann der Sketch getestet werden.

So funktioniert's

In der Funktion *setup* wird die serielle Schnittstelle auf 115.200 Baud konfiguriert. Bei 16-MHz-Quarztakt ergibt sich bei dieser Baudrate zwar ein relativ großer Fehler, dieser wirkt sich jedoch kaum noch auf die Datenübertragung aus. Der Digitalpin 6 wird als PWM-Ausgang verwendet. Anschließend wird der Timer 0 des Arduinos für 8-bit-Auflösung konfiguriert. So ergibt sich ein PWM-Signal mit einer Frequenz von 16 MHz/256 = 62,5 kHz. Der Wert von OCR0A bestimmt das Puls-/

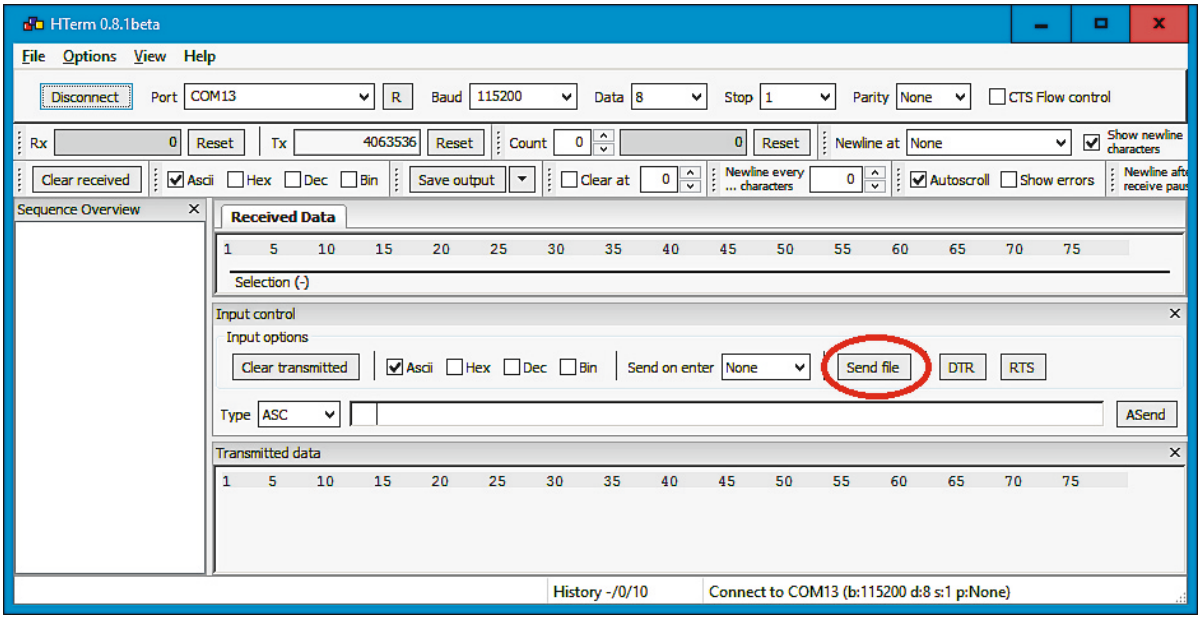

Bild 3: Die Einstellungen im Terminalprogramm

Pausenverhältnis. Weitere Details zu Konfiguration von Timern finden sich in [3 oder 4].

In der Main-Loop wird zunächst mittels einer for-Schleife gewartet, bis 44 Bytes über die serielle Schnittstelle empfangen wurden, da eine WAV-Datei einen 44 Bytes großen Header mit Dateiinfos aufweist. Würden diese Bytes auf den PWM-Ausgang übertragen, könnte dies Störgeräusche im Lautsprecher verursachen.

Anschließend wird in einer while(1)-Endlosschleife jedes über die serielle Schnittstelle empfangene Datenbyte in ein Timer-Register geschrieben und so auf den PWM-Ausgang ausgegeben. Wenn ein kontinuierlicher Datenstrom mit 115.200 Baud empfangen wird, werden bei 10 Bit (1 Startbit, 8 Datenbits, 1 Stoppbit) also 11.520 Bytes pro Sekunde ausgegeben, was in etwa 11 kHz, also der WAV-Datenrate, entspricht. Der Lautsprecher selbst wirkt wie ein Tiefpass und filtert den größten Teil der PWM-Frequenz von 62,5 kHz heraus.

Wiedergabe von Sound-Dateien

Die Wiedergabe von Klangdateien über den PWM-Ausgang des Arduinos stellt natürlich nur eine sehr einfache Möglichkeit dar, den Arduino als Tonwiedergabemedium einzusetzen. Neben der Übertragung über die serielle Schnittstelle könnten die Tondaten auch im Flash des Arduinos gespeichert werden. Mit 16 Kilobyte kann man auf diese Weise allerdings gerade mal drei bis vier Sekunden lange Audiodateien speichern. Zwar könnte man die Speicherkapazität mit einer externen SD-Karte erheblich erweitern, wenn man aber schon zusätzliche Hardware einsetzt, kann auch ein wesentlich effizienteres Tonwiedergabeverfahren verwendet werden.

Genau hier kommt das MP3-Verfahren zum Zug, welches die verlustbehaftete Kompression digital gespeicherter Tondaten erlaubt. MP3 verwendet dafür verschiedene Methoden der Psychoakustik. So sollen nur die relevanten, also für den Menschen wahrnehmbaren Signalanteile abgespeichert werden. So wird bei kaum verringerter Audioqualität eine starke Reduktion des Datenumfangs ermöglicht.

Eine häufig verwendete Datenrate ist 192 Kilobit/Sekunde. Damit kann bereits eine hohe Wiedergabequalität erreicht werden. Die Datenkompression einer MP3-Audiodatei beträgt dabei bereits etwa 85 % gegenüber einer konventionellen Audio-CD. MP3 ist mittlerweile das am weitesten verbreitete Verfahren zur Speicherung und Übertragung von Musik auf Computern, Smartphones und im Internet. Entwickelt wurde das Format ab 1982 am Fraunhofer-Institut für Integrierte Schaltungen in Zusammenarbeit mit der Friedrich-Alexander-Universität Erlangen-Nürnberg, den AT&T Bell Labs und der Firma Thomson.

Bereits Mitte der 1990er-Jahre waren Abspielgeräte und Software für PCs im Umlauf, die es ermöglichten, komprimierte MP3-Dateien zu speichern und abzuspielen. Auch der Austausch solcher Dateien über das Internet vereinfachte sich. Selbst bei einfacher ISDN-Geschwindigkeit benötigte man für die Übertragung lediglich das Zwei- bis Dreifache der Abspielzeit. Mit DSL-Leitungen liegt die Übertragungsdauer sogar weit unterhalb der Spieldauer. Ab 1998 erschienen im Handel die ersten tragbaren MP3-Player.

Mit einem Audio-Shield steht seit einiger Zeit auch eine Möglichkeit zur Verfügung, MP3-Dateien Arduino-gesteuert wiederzugeben. Der besondere Reiz dieser Möglichkeit besteht darin, dass man nun in der Lage ist, MP3-Spieler nach eigenen Vorstellungen aufzubauen. Am Ende dieses Artikels wird dazu ein spezieller MP3-Wecker vorgestellt, der in weitem Rahmen individuell gestaltet und variiert werden kann.

Die Funktionen des Audio-Shields

Zunächst sollen hier aber die wesentlichen Funktionen des Shields (siehe Bild 5) vorgestellt werden.

Wie bei Arduino-Anwendungen üblich, wird zum Audio-Shield eine Library zur Verfügung gestellt. Diese kann von der Website des ELV Shops geladen werden [6].

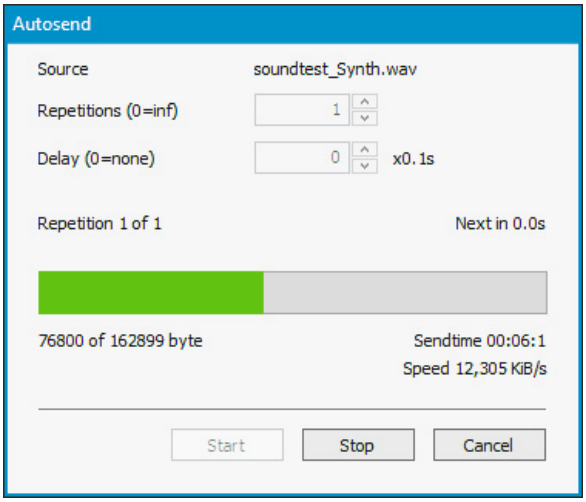

Bild 4: HTerm sendet Audiodaten

Diese Library enthält bereits einige einfache Beispiele, welche die Funktionen des Audio-Shields erläutern. Die wichtigste Funktion ist das Abspielen einer MP3-Datei von einer SD-Karte. Der Decoder wird entsprechend über die Library VS10xx angesprochen, dabei muss diese am Anfang mit *VS1011.begin()* initialisiert werden. Über *VS1011.Reset()* wird der MP3-Decoder zurückgesetzt, dann können die Daten in 32-Byte-Paketen übertragen werden.

Um die Audiosignale hörbar zu machen, muss entsprechend die Mute-Schaltung deaktiviert werden. Damit wird auch gleichzeitig der Verstärker aktiviert.

Die Library des Audio-Shields stellt für die Ansteuerung des MP3-Decoders das Objekt „VS1011" bereit, welches die folgenden Funktionen enthält. Dabei sind den Funktionen die Objektnamen mit einem Punkt voranzustellen, beispielsweise zur Initialisierung der Library:

VS1011.begin();

Der auf dem Shield integrierte Verstärker kann über eine Mute-Funktion gesteuert werden. Über

void UnsetMute(void);

wird der Verstärker aktiviert, über

void SetMute(void);

kann er deaktiviert werden.

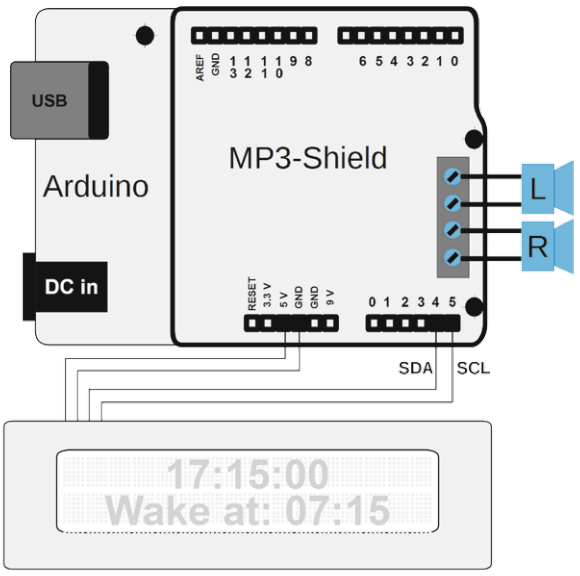

Bild 5: Shield und LC-Display am Arduino

Die eigentliche Initialisierung des Audio-Shields erfolgt über

void begin(void);

des MP3-Decoders. Zwischen zwei Audiodateien sollte jeweils ein Software-Reset ausgeführt werden:

void SoftReset(void);

Ein besondere Vorzug des Shields ist, dass die Lautstärke über die Funktion

void SetVolume(unsigned char leftchannel, unsigned char rightchannel);

programmgesteuert einstellbar ist. Zu beachten ist allerdings, dass die maximale Lautstärke beim Einstellwert 0, die minimale Lautstärke bei 254 erreicht wird. Über den Wert 255 kann die Ausgangsstufe vollständig deaktiviert werden. So kann man etwa über zwei Taster die Lautstärke einstellbar machen. Ein analoges Potentiometer ist dann nicht mehr erforderlich.

Aber auch bei einem Wecker kann man so zunächst mit geringer Lautstärke beginnen und diese dann langsam steigern. Mit der Anweisung

unsigned char Send32(unsigned char* pBuffer);*

wird ein 32-Byte-Datenblock an den MP3-Decoder übertragen. Der Puffer des MP3-Decoders kann über

void Send2048Zeros(void);

geleert werden.

Um die SD-Karte nutzen zu können, muss die SD-Library des Arduinos hinzugefügt werden, dabei ist zu beachten, dass der ChipSelect der SD-Karte beim Audio-Shield von der SD-Library abweicht und deshalb beim Initialisieren diese als Parameter zu übergeben ist:

SD.begin(SD_CS);

Damit ergibt sich das folgende Minimalprogramm zum Abspielen der Datei 001.mp3:

```
// play "001.mp3" from SD-card

#include <SD.h>
#include <SPI.h>
#include <AudioShield.h>

void setup()
{ SD.begin(SD_CS);
  VS1011.begin();
  VS1011.SetVolume(0,0);
}

void loop()
{ unsigned char buffer[32];
  if( File SoundFile = SD.open("001.mp3", FILE_READ))
  { VS1011.UnsetMute();
    while(SoundFile.available())
    { SoundFile.read(buffer, sizeof(buffer));
      VS1011.Send32(buffer);
    }
    VS1011.Send2048Zeros();
    VS1011.SetMute();
    SoundFile.close();
  }
}
```

Wie die Register des MP3-Decoders im Einzelnen ausgelesen oder beschrieben werden, kann einem technischen Artikel in einer früheren Ausgabe des ELV Journals entnommen werden [6]. Weitere Detailinformationen sind auch dem im Datenblatt des auf dem Shield verwendeten Bausteins VS1011 der Firma VLSI Solution zu finden.

Zudem befinden sich auf dem Shield noch mehrere LEDs, die softwaregesteuert ein- oder ausgeschaltet werden können. Für den Betrieb der LEDs sind verschiedene Wischverbinder auf dem Shield zu schließen. Allerdings sind die LEDs nicht sichtbar, sobald der Aufbau in ein Gehäuse eingebaut wird. Deshalb wird die Verwendung dieser LEDs hier nicht weiter betrachtet. Bei entsprechendem Interesse können Details dazu wiederum in [6] nachgelesen werden.

Nachdem eine MP3-Datei auf die SD-Karte kopiert wurde und die Karte in den Slot des Shields eingesteckt ist, steht einem ersten Test nichts mehr im Weg. Man muss lediglich noch zwei Lautsprecher an die Schraubklemmen anschließen und nach dem Laden des Sketches wird die Datei abgespielt.

Tipp:

Die Verwendung des Audio Shields sollte mit einer externen Stromversorgung am Arduino erfolgen, da es sonst bei größeren Lautstärken zu Problemen mit der USB-Stromversorgung kommen kann.

Anschluss von Aktivboxen

Auf dem MP3-Shield ist zwar ein Audioverstärker vorhanden, dieser liefert jedoch nur eine relativ geringe Ausgangsleistung. Für Kopfhörerbetrieb ist diese durchaus ausreichend. Will man jedoch Lautsprecher anschließen, bietet sich der Einsatz von Aktivboxen an. Diese können direkt mit den Schraubklemmen verbunden werden. Für eine optimale Klangqualität ist in diesem Falle der Lautstärkepegel entsprechend anzupassen, sodass es nicht zu Übersteuerungen kommt. Natürlich kann man sich geeignete Verstärker auch selbst bauen. Hinweise zum Aufbau eines einfachen Stereoverstärkers finden sich z. B. in [7].

Musikwecker

Lässt man sich von einem gewöhnlichen Radiowecker aus dem Schlaf holen, hat das verschiedene Nachteile. Häufig wird man dann nicht von angenehmer Musik geweckt, sondern von nervigen Werbeeinblendungen, die sich im Halbschlaf ja besonders gut einprägen sollen.

Außerdem verfügen kommerzielle Radiowecker meist nur über eine einzige Weckzeit. Deutlich komfortabler wäre es doch, wenn man für jeden Tag eine andere Zeit einprogrammieren könnte. So wäre für Wochentage beispielsweise 7:00 Uhr angemessen, während man am Samstag oder Sonntag erst um 9:00 Uhr geweckt werden will.

Hier kommen die Vorteile eines Eigenbaus zum Tragen. Mit dem MP3-Shield und einem Arduino kann man sich alle oben genannten Wünsche problemlos erfüllen. Man muss sich nur seine Lieblingssongs auf die SD-Karte kopieren und kann sich dann ganz individuell mit seinen Lieblingsliedern wecken lassen.

Für vollen Komfort: Display für Uhr- und Weckzeit

Natürlich gehört zu einem ordentlichen Wecker auch die Anzeige der Uhr- und Weckzeit. Hierfür kann ein Display mit dem Arduino verbunden werden. Da das MP3-Shield allerdings bereits eine Vielzahl von Pins belegt, ist der Anschluss eines klassischen HD44780-Displays, wie es in Artikel „Alphanumerische LC-Displays" im ELV Journal 4/2016 vorgestellt wurde, mit erheblichen Problem verbunden. Eine Lösung bietet die Verwendung eines „I²C"-Displays, das mit nur zwei aktiven Pins gesteuert werden kann. Zudem bleiben auch bei aktivem Shield die beiden Pins für den I²C-Bus frei, sodass dieser Displaytyp hier ohne Zusatzaufwand verwendet werden kann. Bild 6 zeigt den zugehörigen Aufbau.

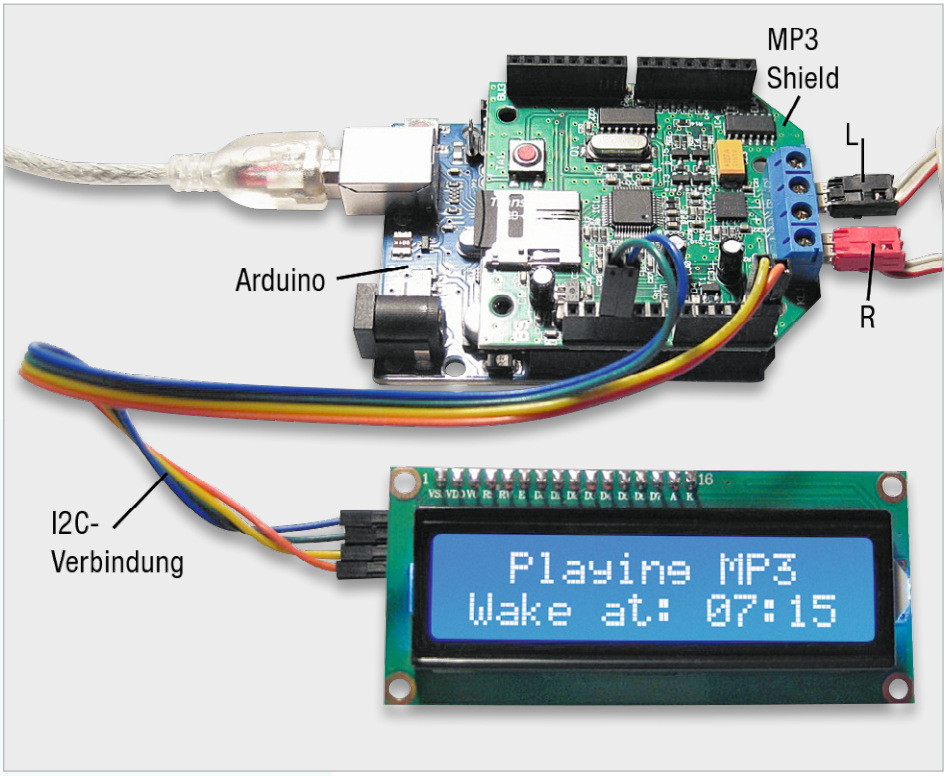

Bild 6: Aufbau mit MP3-Shield, LC-Display und Arduino

Der Sketch dazu sieht so aus:

```
// MP3Clock.ino

#include <Time.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SD.h>
#include <SPI.h>
#include <AudioShield.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

int startHour = 7;
int startMinute = 15;

void printDigits(int digits)
{ // print leading zeros
  if(digits < 10) lcd.print('0');
  lcd.print(digits);
}

void setup()
{ lcd.begin();  lcd.backlight();
  lcd.setCursor(0, 1); lcd.print(" Wake at: ");
  printDigits(startHour); lcd.print(":"); printDigits(startMinute);
  // 07:14:45 am 1. Januar 2017.
  setTime(07, 14, 45, 01, 01, 2017);
  // setup MP3 shield
  if( SD.begin( SD_CS ) == false )
  { return; }
  VS1011.begin();
}

void loop()
{ // show current time
  lcd.setCursor(4, 0); printDigits(hour()); lcd.print(":");
   printDigits(minute()); lcd.print(":"); printDigits(second());

   if ((hour() == startHour && minute() == startMinute))   // Start MP3
    { lcd.setCursor(0, 0); lcd.print("  Playing MP3   ");
      unsigned char buffer[32];     // generate buffer
      if( File SoundFile = SD.open( "001.mp3", FILE_READ ) )
      { VS1011.UnsetMute();         // power up amplifier
        while( SoundFile.available() )
        { SoundFile.read( buffer, sizeof(buffer) );
          VS1011.Send32( buffer );
        }
        VS1011.Send2048Zeros();     // fill buffer
        VS1011.SetMute();           // amplifier off
        SoundFile.close();
      }
      lcd.setCursor(0, 0); lcd.print("                ");
    }
}
```

Bild 7: Das Display zeigt die aktuelle Zeit und den Systemstatus.

Bild 8: Der MP3-Wecker mit Display und Aktivboxen

Zusätzlich zu den bereits genannten Bibliotheken sind noch die folgenden Libs erforderlich:

- Time-Library
- Wire.h-Library
- LiquidCrystal_I2C-Library

Die Bibliotheken können wie üblich kostenfrei aus dem Internet geladen werden. Informationen zu den jeweiligen Links finden sich am Ende des Artikels. Wenn man den Arduino samt dem aufgesteckten MP3-Shield in ein formschönes Gehäuse einbaut, dann steht dem morgendlichen Musikgenuss nichts mehr im Weg. Die Bilder 7 und 8 zeigen einen entsprechenden Aufbauvorschlag dazu.

Ausblick

Mit diesem Artikel ist der Ausflug in die Klangerzeugung und Musikwiedergabe mit dem Arduino beendet. Mit dem Anschluss des I²C-Displays wurde jedoch bereits das nächste Thema gestreift. Wie man bereits erkennen konnte, kann man durch den Einsatz von Bussen Pins und andere Ressourcen in einem Mikrocontrollersystem einsparen.

So ist es möglich, die Funktionalität des Arduinos oder von Mikrocontrollersystemen allgemein durch Bussysteme wie I²C oder SPI ganz erheblich zu erweitern. Auch der SPI-Bus kam ja bereits bei der Ansteuerung der SD-Karte im Hintergrund zum Einsatz.

Aus diesem Grunde werden sich die nächsten Artikel genauer mit den bei Mikrocontrolleranwendungen so beliebten Bussystemen wie

- I2C
- SPI
- One-Wire

befassen.

Wie immer werden dabei neben den theoretischen Hintergründen auch praktische Anwendungen vorgestellt. Insbesondere die Anbindung von Sensoren und Aktoren an die verschiedenen Busse wird dabei eine zentrale Rolle spielen. ELV

Download-Paket zum Artikel:

Die Sketche und Beispieldateien zu diesem Artikel können unter www.elv.de: Webcode #10071 heruntergeladen werden.

Weitere Infos:

[1] Audacity findet sich unter: http://www.audacityteam.org/download/
[2] Das Terminalprogramm kann aus dem Internet geladen werden unter: http://www.der-hammer.info/terminal/
[3] Mikrocontroller-Onlinekurs, Franzis-Verlag, exklusiv für ELV, 2011, Best.-Nr. CM-10 20 44
[4] Elektor Praxiskurs AVR-XMEGA-Mikrocontroller Best.-Nr. CM-12 07 62
[5] G. Spanner: Arduino – Schaltungsprojekte für Profis, Elektor-Verlag, 2012
[6] Artikel „ Audio Shield for Arduino“ im ELV Journal 1/2013
[7] Im E-Book „Audiotechnik“ (siehe https://www.amazon.de/dp/B013NSPPY6) finden sich Grundlagen zum Bau von Audioverstärkern

Preisstellung Februar 2017 – aktuelle Preise im Web-Shop

Empfohlene Produkte	Best.-Nr.	Preis
Arduino UNO	CM-10 29 70	€ 27,95
Audio Shield	CM-11 06 48	€ 9,95

Alle Arduino-Produkte wie Mikrocontroller-Platinen, Shields, Fachbücher und Zubehör finden Sie unter: www.arduino.elv.de