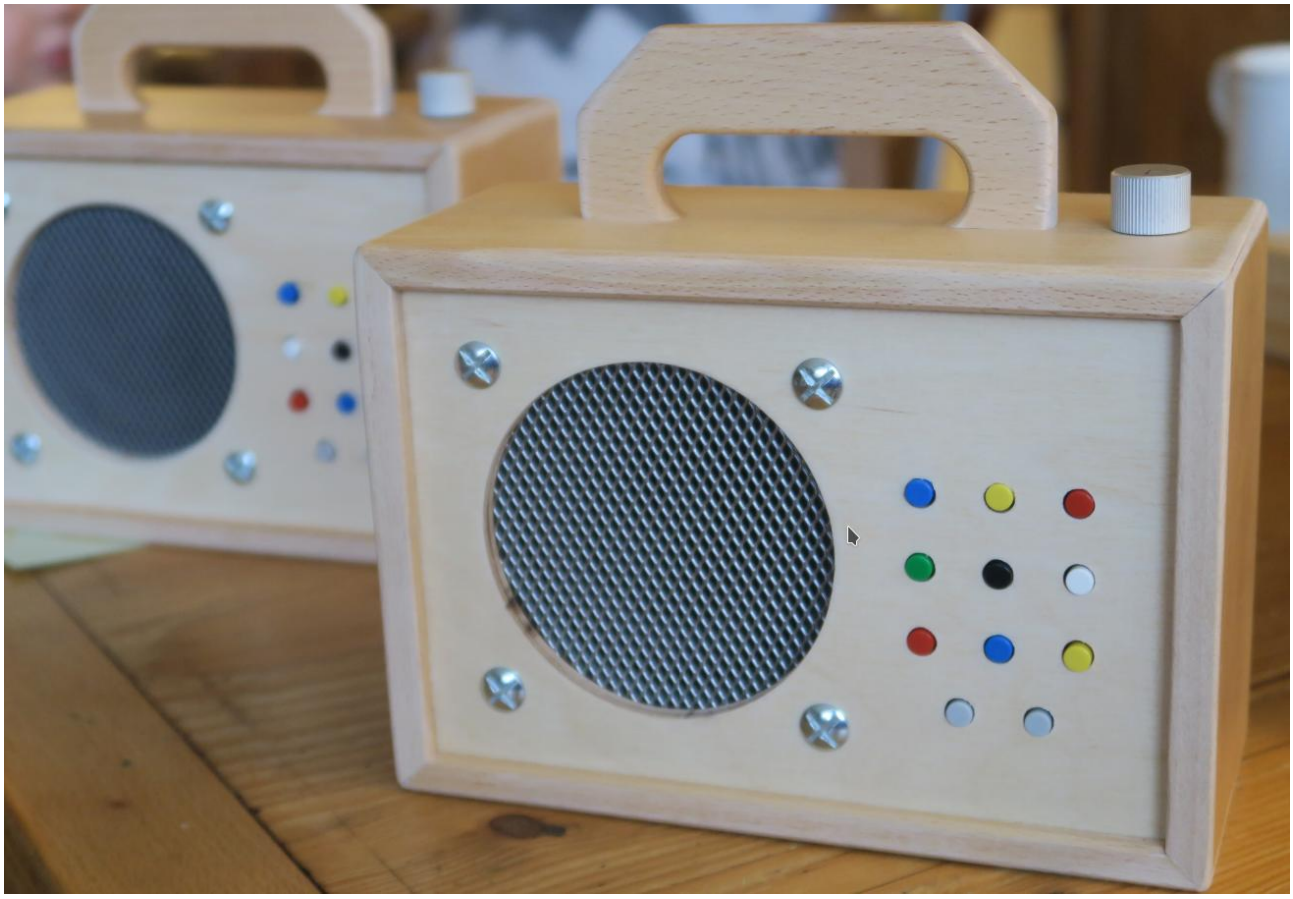
Konzept für Kleinkind Musik-Spieler

Version vom **05.01.20 23:12:43**



1 Übersicht

Bau eines Kleinkind Musik-Abspiel-Gerätes nach Inspiration durch den „Hörbert“ (https://www.hoerbert.com).

Die Kinder sollen ein einfach zu bedienendes und schönes Gerät haben, mit dem sie Musik, Hörspiele usw. von einer SD-Karte abspielen können.

Die Bedienung durch die Kinder soll auf das wesentliche reduziert sein: An/Aus, Lautstärke, Titel-Wahl, zurückspulen, vorspulen, ...

Erweiterte Einstellungen sowie Wechsel der Batterien und SD-Karte erfolgt durch die Eltern über ein „Eltern-Fach“ auf der Rückseite, das verschlossen ist.

Musikausgabe erfolgt über einen Lautsprecher (Mono). Somit hören die Kinder und Eltern zusammen, die Eltern haben Kontrolle, was die Kinder hören und die Kinder werden nicht „abgestellt“.

Damit das Gerät transportabel ist, erfolgt die Stromversorgung über (aufladbare) Batterien. Standardtyp ist AA (Mignon) für möglichst hohe Portabilität und Sicherheit (kein Liion).

Die SD-Karte wird mit WAV/PCM Dateien bespielt (nicht MP3); das Abspielen von PCM ist viel stromsparender als MP3-Dekodierung, somit steigt die Akku-Laufzeit.

Die Konvertierung vorhandener Musik/CDs und das Bespielen erfolgt am PC durch die Eltern.

Das Gehäuse besteht aus robustem Holz, z.B. Multiplex oder MassivBuche etc.

Technologische Basis ist ein Arduino (Pro Mini). Der ATMega328 ist mehr als ausreichend, läuft mit 5V (= 4xAA Batterie/Akku) und kann frei programmiert werden.

Die Software ist eine Eigenentwicklung und wird unter GLPL veröffentlicht.

https://github.com/ludgerknorps/kimubo

Inhaltsverzeichnis

2 Einleitung

Dieses Dokument ist in Arbeit.

Dinge, die zwar prinzipiell vorgesehen sind, aber noch nicht fertig definiert sind UND die für den wesentlichen Betrieb erstmal nicht notwendig sind, sind in PINK markiert. Pinke Umfänge sind in späteres Ausbaustufen umsetzbar.

Dinge, die für den wesentlichen Betrieb notwendig sind und noch nicht fertig definiert oder fehlerhaft sind, sind in ROT markiert.

Dinge, die für den wesentlichen Betrieb notwendig sind und prinzipiell fertig definiert sind aber noch geprüft/diskutiert werden müssen, sind in GELB markiert.

3 Linksammlung

- 20 Mhz Arduino: https://tttapa.github.io/Pages/Arduino/Bootloaders/ATmega328P-custom-frequency.html
- Optimieren SRAM: https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram
- Sehr gute Erklärung PWM (FastPWM): https://arduino-projekte.webnode.at/registerprogrammierung/fast-pwm/
- dual pwm outputs → 16bit: http://www.openmusiclabs.com/learning/digital/pwm-dac/dual-pwm-circuits/index.html
- Fuse Settings for ATMEGA328: http://www.martyncurrey.com/arduino-atmega-328p-fuse-settings/

4 E/E-Architektur

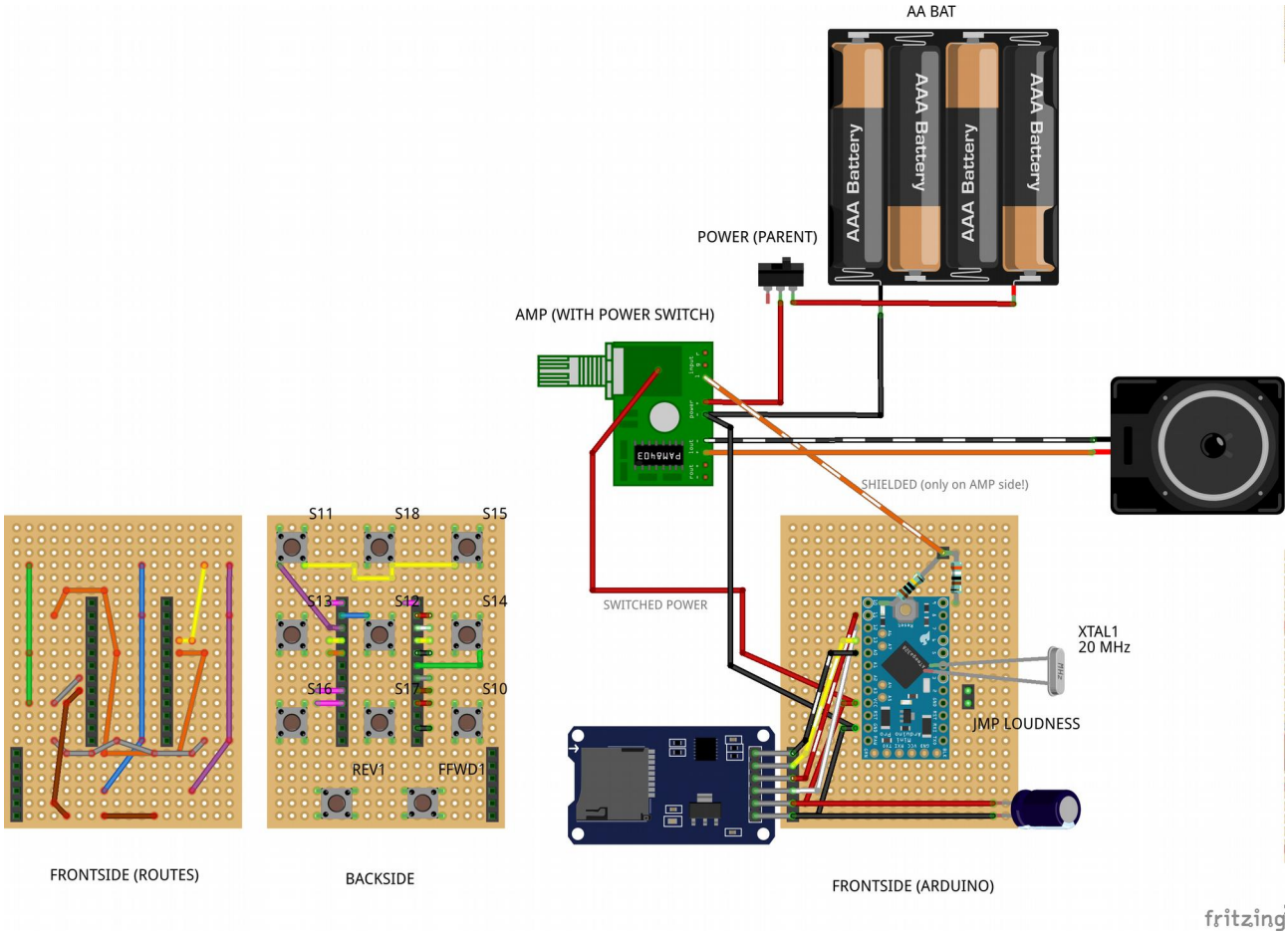


Schaubild 1: Elektronik des KIMUBO; aufgebaut auf Lochraster-Platine

4.1 Subsysteme/Komponenten

1 Arduino / Prozessor

Zentraler Rechner, der Tastatur und andere Inputs auswertet, Daten aus SD-Karte liest und an Audio-Teil übergibt. Ausgabe der Audio-Daten erfolgt als PWM-Analog-Signal.

2 Daten-Teil

Daten auf SD-Karte liegen als WAV/PCM formatierte Audiodaten vor (16bit, Mono, 32kHz Samplingrate). Werden von Arduino gelesen.

3 Audio-Teil

Besteht aus einem (Mono)Verstärker, der das vom Arduino gelieferte PWM-Audio-Signal verstärkt.

Am Verstärker ist ein Lautstärke-Poti angeschlossen, mit dem von außen (das Kind) die Lautstärke einstellen kann.

Wenn die Lautstärke auf „ganz leise“ gestellt ist, wird das Abspielen automatisch angehalten (Pause). Wenn Lautstärke erhöht wird, wird automatisch weitergespielt (resume).

Hinter der Eltern-Klappe befindet sich ein Laufstärker-Vorwähler (Schalter/Jumper), mit dem die Eltern eine generelle Laut/Leise-Vorwahl treffen können (das Kind kann dann mit dem normalen Lautstärke-Poti nur innerhalb des vorgewählten Lautstärke-Niveaus regeln). Diese

1bit Information wird vom Arduino verwendet um die Amplitude des PWM-Audio-Signals zu berechnen (hat also nichts mit Verstärker zu tun).
Tonausgabe erfolgt über einen Lautsprecher im Gehäuse.

Das auf dem PAM8403 bestehende AMP-Modul muss modifiziert werden, um eine kindgerechte Maximallautstärke zu erreichen, siehe Kapitel

4 Tastatur

Über die Tastatur kann das Kind wählen, welches „Werk" abgespielt werden soll (9 mögliche). Entsprechend gibt es 9 farbige Taster zur Direktwahl.

Zusätzlich gibt es zwei graue Taster für Lied-zurück/zurückspulen und Lied-vor/vorspulen. Lied vor/zurück erfolgt durch kurzen Tastendruck, vorspulen/zurückspulen erfolgt durch langen Tastendruck. Letzteres bedeutet 10 sek. vor oder zurück.

5 Stromversorgung

Die Stromversorgung erfolgt über 4x 1,5V AA Batterien/Akkus (Mignon-Zellen). Die Zellen müssen zum Laden entnommen werden. Dadurch entfällt zum Einen eine Ladebuchse und Ladeschaltung. Zum anderen kann nahtlos ohne Kabelanschluss weitergehört werden (neue Akkus einsetzen).

Hinter der Eltern-Klappe befindet sich der Eltern-An/Aus-Schalter, mit dem das ganze Gerät stromlos geschaltet wird.

Oben am Gerät befindet sich – integrierte in das Lautstärke-Poti – ein An/Aus-Schalter, mit dem die Kinder das Gerät aktivieren können (nur, wenn Eltern-An/Aus-Schalter auf „an" steht). Dieser schaltet die Stromversorgung zum Arduino.

StromSparen:

Der Kimubo benötigt bei maximaler Lautstärke ca. 60..70mA. Um Energie zu sparen und die Batterie-Laufzeit zu erhöhen, werden in einer Ausbaustufe PowerSave-Modes eingeführt.
Die drei Hauptverbraucher sind: a) der Arduino selbst, b) der AMP, c) die SD-Karte mit Adapter-Platine.
Nach Ende einer Playlist plus einer Wartezeit wird der Arduino den AMP mittels Suspend-Funktion abschalten (LogikSignal) und selbst in SLEEP_MODE_PWR_DOWN gehen. Daraus wacht er nur durch Power-Off-On wieder auf.[1]
Die SD-Karte geht nach Aussagen in https://forum.arduino.cc/index.php?topic=149504.0 automatisch in einen hinreichenen Energiesparmodus.

6 Debug-Subsystem

Updates für die Arduino Software können über den ISPC (In-Service-Programming-Connector) erfolgen. Dazu ist ein FTDI USB to Serial/UART TTL Adapter an den ISPC anzuschließen. Dieser befindet sich auch hinter der Eltern-Klappe.

Ebenso ist über den ISPC auch ein Debugging möglich:

In der Software kann mittels eines zentralen #define debug Preprozessor-Befehls der Debug-Mode aktiviert werden. Die Software muss dann neu auf den Arduino geflasht werden. Aktiviertes Debug verbraucht mehr Strom, und ist daher gesondert zu aktivieren und im Normalfall nicht aktiv.

Neben dem ISPC befinden sich hinter der Eltern-Klappe auch noch ein Status-Taster: wenn der Taster gedrückt wird, "erzählt" das Gerät seinen aktuellen (Fehler-)Status inkl. der aktuellen Batteriespannung.

1 Noch später kann ein Tastendruck einen Aufweck-Interupt auslösen, siehe http://ww1.microchip.com/downloads/en/AppNotes/doc1232.pdf

5 16bit PWM-Audio Ausgabe zum Verstärker

Wir benutzen den PWM-Generator im Arduino, um „quasi-analoge“ Signale zu erhalten.

Wesentliche Parameter sind

- die Sample-Rate (auch: Wiederholrate) – in 1 pro Sekunde: wie oft pro Sekunde wird ein neuer analoger Wert am Ausgang bereitgestellt?
  Je höher, desto besser ist die Audio-Qualität, weil schnelle Signaländerungen nicht durch zu wenige Ausgaben pro Sekunde „verwaschen“ werden. Je höhere Frequenzen noch gut hörbar sein sollen, desto höher muss auch die Sample-Rate sein.
  Faustformel: höchste Frequenz * 2 = Sample-Rate.
- Die Auflösung – in bit: wie viele unterschiedliche Lautstärkestufen sind im analogen Wert am Ausgang unterscheidbar?
  Je höher desto besser ist die Audio-Qualität, insb. bei leisen Passagen[2]

Die PWM-Signale müssen allerdings auch vom relativ schwachbrüstigen Arduino erzeugt werden können.

5.1 16bit PWM-Audio mit Timer/Counter1

Wir verwenden im Arduino den TIMER/COUNTER1, der auch 16bit PWM unterstützt.

Damit die Soundqualität gut ist und zur einfacheren Berechnung bei Audioausgaben gilt:

1. Sample-Rate der PWM == Sample-Rate der auszugebenden PCM-Datei
2. Die Auflösung der PWM >= Auflösung der auszugebenden PCM Datei.

Eine 16bit 16kHz PCM Datei könnte somit mit einem 16bit PWM-Ausgang mit 16kHz PWM-Sample-Rate ausgegeben werden.

Allerdings schafft der Arduino ATMEGA328P bei 16bit aufgelöster PWM nur maximal eine PWM-Wiederholrate von F_CPU / 2^16. Also bei den typischen 16MHz nur max. 16Mhz/65536 = 244Hz Wiederholrate. Das ist weit weg von den 16kHz, die nötig wären.

Reduzieren der Auflösung hilft: bei 8bit schafft er 16Mhz/256 = 62500Hz.

Das würde sogar für 4x Oversampling reichen – aber eben nur mit 8 bit Auflösung. Insbesondere schlecht für leise Hörspiele wie für Kinder…

Lösung:

Man kann auch zwei 8bit PWM „hintereinanderschalten“ und erhält auch die volle 16bit Auflösung bei der Wiederholrate einer 8bit PWM.

Um die beiden Arduino-Ausgangsports zu einem 16bit Ausgang zu verschalten, ist es notwendig, zwei 8bit DA-Ausgänge zusammen zu verwenden. D9 im folgenden Bild ist der Ausgang für die „oberen“ 8bit und D10 ist der Ausgang für die „unteren“ 8bit. Zusammen dann 16bit[3].

Die Signalamplitude der unteren 8bit muss genau ein 256tel der Signalamplitude der oberen 8bit sein. Dies erfolgt mittels einer R-R-Leiter („Widerstandsleiter“) aus einem 3k9 und einem 1M Widerstand. Warum diese Werte? Weil wir an D9 ein genau 256 mal so großes Signal anliegen haben

2 8bit heißt, dass das analoge Signal in 256 Lautstärkestufen unterteilt wird. 16Bit heißt, dass es 65536 Lautstärkestufen sind. Der hörbare Unterschied zwischen 8bit und 16bit ist im Wesentlichen, dass bei leiserem Signal bei 8bit ein deutliches Rauschen vernehmbar ist. Bessere Tonqualität hat somit ein 16bit-Signal. Bei leisem Signal – wenn also das Signal z.B. sowieso nur ein 10tel der möglichen Amplitude nutzt – ist das 8bit Signal somit nur sehr schlecht auflösend → wir hören das als Rauschen. Soundtest siehe z.B. https://www.audiocheck.net/audiotests_dithering.php

3 Wenn man den Verstärker nur an D9 anschließt und der Arduino eine 16bit-kodierte Datei abspielt, dann hört man das erwartete Audiosignal – allerdings etwas verrauscht, da ja die unteren 8bit des Signals fehlen. Wenn man den Verstärker nur an D10 anschließt hört man nur „Rauschen“. Wenn aber beide Signale richtig übereinandergelegt werden (s.o.), dann verschwindet das Rauschen (Magie!) und man hört ein schönes unverrauschtes Audiosignal.

wollen wie an D10.

1M / 3,9k = 256,4 ziemlich gut!

Wichtig ist noch, dass wir 1% Widerstände verwenden, die Streuung sorgt sonst für schlechtes Audio.

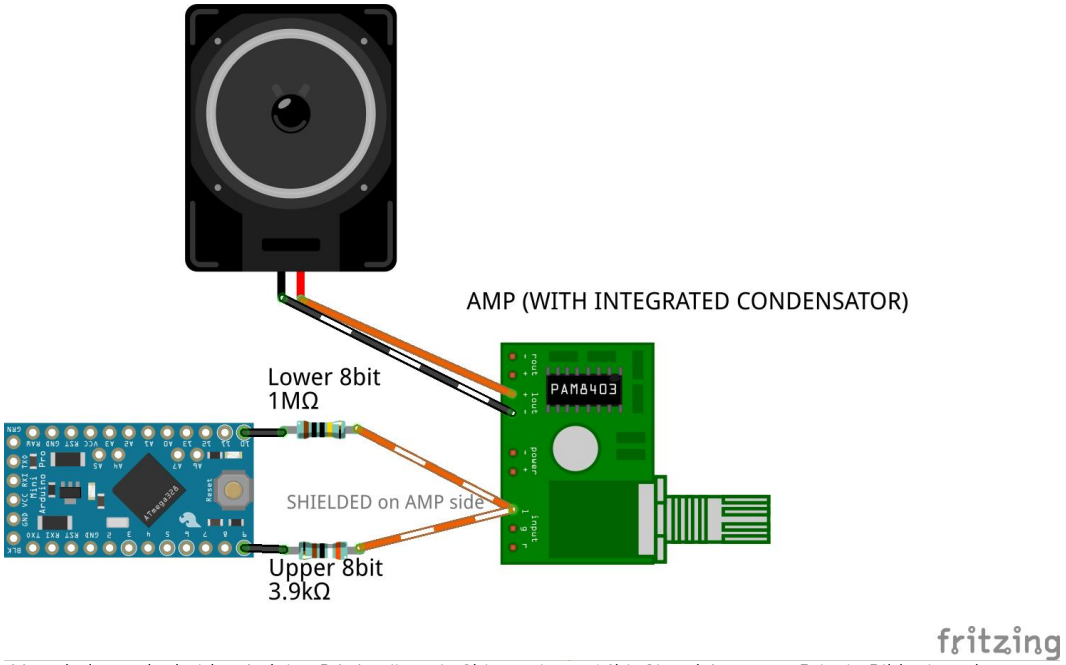


Schaubild 2: Verschaltung der beiden Arduino DA-Ausänge je 8bit zu einem 16bit Signal-Ausgang. Prinzip-Bild mit anderen Widerstandswerten.

Link zu „PWM auf dem ATMEga328: https://sites.google.com/site/qeewiki/books/avr-guide/pwm-on-the-atmega328

5.2 Preisfrage: warum kann der Counter-Wert, bis zu dem der PWM-Counter zählt, einen größeren Wert haben als 2^Auflösung?

Hier sehen wir ein BSP mit Mode 8 für Counter/Timer1 PWM (siehe Datenblatt ATMEGA328P DS40002061A-page 141).

Counter-TOP Wert von ICR1 = 258 entsprechend 31kHz Samplerate – s. Bild.[4]

Aktueller „Ausgabewert“ (also das, was wir an Byte von einer PCM Datei aktuell ausgeben wollten) ist 180.

Das Maximum wäre 255 (8bit Auflösung), das Minimum 0.

Der Counter1 zählt also immer bis 258 hoch, der Dutycycle ist max. 255/258.

Das geht, weil: wir wollen am Ausgang ein analog Signal erzeugen, d.h. das PWM-Ausgangssignal wird sowieso noch tiefpass-gefiltert. D.h. auch, dass der absolute analog-Pegel nicht interessiert. Es ist ausreichend, dass das Audio-Signal in 8bit Auflösung ausgegeben wird:

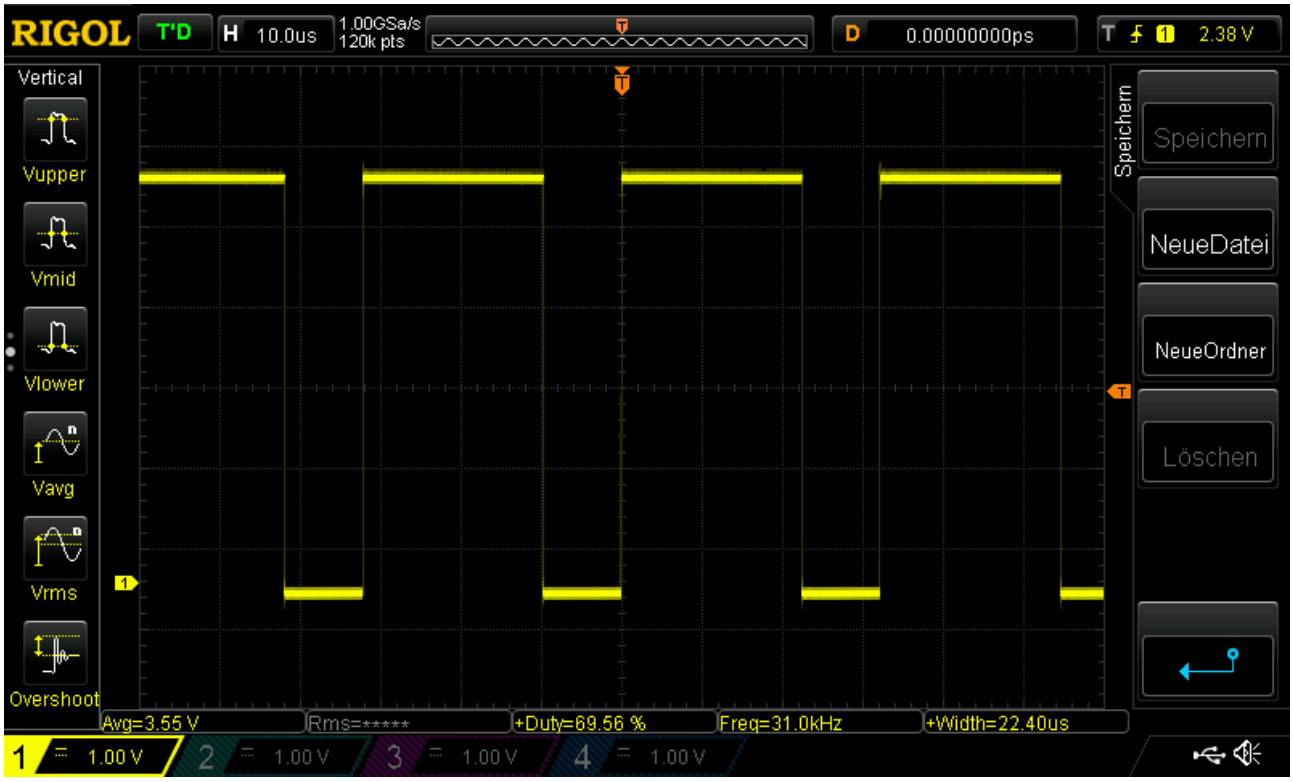
- aktueller PCM-Audio-Wert = 0 → Dutycycle = 0 → Analogsignal = 0V
- aktueller PCM-Audio-Wert = 255 → Dutycycle = 255/258 → Analogsignal = <max>
- aktueller PCM-Audio-Wert = 180 → Dutycycle = 180/258 → Analogsignal = 180/258*<max>

4 16.000.000 Mhz / 31.000 / 2 = 258,06
Hinweis: Mode 8 PWM ist sogenannte Phase-and-frequency-correct PWM --> d.h. dass wir "symmetrische" PWM-Signale erzeugen, wodurch die Audioqualität nochmals verbessert wird (Symmetrie == weniger Oberwellen). Dies erfolgt zum Preis einer halbierten maximalen Samplerate. Daher Faktor "/2" oben.

D.h. je kleiner die gewünschte Auflösung im Vergleich zu Counter1-TOP-Wert ist, umso geringer wird die gesamte Ausgangsamplitude des Analog-Signals. Aber die Auflösung bleibt immer gleich groß – es lassen sich immer gleich viele Lautstärkeabstufungen innerhalb der Ausgangsamplitude unterscheiden.

Je näher 2^Auflösung an TOP liegt, umso lauter das Signal.

Da die Sample-Rate nicht innerhalb eines Stücks wechselt, lassen sich solche Lautstärkeunterschiede über den nachgelagerten Audio-Verstärker ausgleichen.



5.3 32kS/s mittels Arduino hochgetaktet auf 20Mhz

Bei 16Mhz kann der Arduino (ATMega328) im Phase-and-frequency-correct-mode gerade keine echten 32kHz Sampling-Rate unterstützen. Gemäß obiger Rechnung ergibt sich der schnellste Timer-Überlauf bei 16MHz zu

16.000.000Hz / 255 / 2 = 31.372,6Hz.

Bei 20Mhz erhöht sich diese Frequenz auf

20.000.000Hz / 255 / 2 = 39215,7Hz.

Damit sind 32kHz realisierbar.

Nachteile der Takterhöhung sind

1. Aufwand für Umbau des Arduino
2. Aufwand für Neuflashen eines modifizierten Bootloaders in den Arduino und Modifizieren der Arduino-IDE.
3. Etwas höherer Stromverbrauch

4. Etwas instabiler bei geringen Spannungen (unterhalb 4V)
5. Für die (normierte) Samplerate von 32.000Hz ist ein etwas höherer TOP-Wert des Counter/Timer1 nötig:

   20.000.000Hz / 32.000Hz = 312,5 --> 312 oder 313.

   D.h. bei gleichbleibenden 8bit Auflösung eine längere Zeit mit OCRmax = 2^8 = 256 < TOP. Und das bedeutet eine geringere mittlere Signalamplitute, die allerdings, wie oben beschrieben, per Verstärker ausgleichbar ist.

Vorteile sind

1. Samplerate von 32.000Hz möglich
2. mehr Hintergrund-Rechenpower.
   Allerdings: ist diese nötig?
   Ja, wahrscheinlich indirekt schon: für kleinere PCM-Buffer! (siehe nächster Punkt)
3. kleinere Buffer sind möglich.
   31.000Hz Samplerate sind zwar bei 16Mhz möglich, erfordern aber Buffer von >ca. 320 Byte. Der statische Gesamt-RAM-Verbrauch ist somit >75% von den total 2048Byte. Das ist schlecht, weil es Instabilitäten fördert. Außerdem kann es sehr gut sein, dass wir noch RAM für Statemachines, Tastaturabfragen usw. benötigen.
   Somit könnte RAM der entscheidende Faktor werden.
   Bei 20MHz sind deutlich kleinere Buffer möglich, wodurch RAM für andere Tasks frei wird!

Anleitung:
https://tttapa.github.io/Pages/Arduino/Bootloaders/ATmega328P-custom-frequency.html

5.3.1 Veränderungen an Arduino IDE und Bootloader

5.3.1.1 Bootloader

In

```
arduino-1.8.9/hardware/arduino/avr/bootloaders/optiboot/
```

Makefile editieren.

Hinzu:

```
[...]

# sjs 20191114
COMMON_OPTIONS_WITHOUT_BAUDRATE = $(LED_START_FLASHES_CMD) $(BIGBOOT_CMD)
COMMON_OPTIONS_WITHOUT_BAUDRATE += $(SOFT_UART_CMD) $(LED_DATA_FLASH_CMD) $(LED_CMD) $
(SS_CMD)
COMMON_OPTIONS_WITHOUT_BAUDRATE += $(SUPPORT_EEPROM_CMD) $(LED_START_ON_CMD) $(APPSPM_CMD)
COMMON_OPTIONS_WITHOUT_BAUDRATE += $(VERSION_CMD)

# sjs 20191114
COMMON_OPTIONS = $(BAUD_RATE_CMD) $(COMMON_OPTIONS_WITHOUT_BAUDRATE)

[...]

HELPTEXT += "target atmega328*    - ATmega328, ATmega328p, ATmega328pb at 20Mhz!\n"
atmega328_20MHz: TARGET = atmega328
atmega328_20MHz: MCU_TARGET = atmega328p
atmega328_20MHz: CFLAGS += "-DBAUD_RATE=19200 "
atmega328_20MHz: CFLAGS += $(COMMON_OPTIONS_WITHOUT_BAUDRATE)
atmega328_20MHz: AVR_FREQ ?= 20000000L
ifndef BIGBOOT
atmega328_20MHz: LDSECTIONS  = -Wl,--section-start=.text=0x7e00 -Wl,--section-
```

```
start=.version=0x7ffe
else
# bigboot version is 1k long; starts earlier
atmega328_20MHz: LDSECTIONS  = -Wl,--section-start=.text=0x7c00 -Wl,--section-
start=.version=0x7ffe
endif
atmega328_20MHz: $(PROGRAM)_atmega328_20Mhz.hex
ifndef PRODUCTION
atmega328_20MHz: $(PROGRAM)_atmega328_20Mhz.lst
endif

atmega328_isp_20MHz: atmega328
atmega328_isp_20MHz: TARGET = atmega328
atmega328_isp_20MHz: MCU_TARGET = atmega328p
ifndef BIGBOOT
# 512 byte boot, SPIEN
atmega328_isp_20MHz: HFUSE ?= DE
else
# 1k byte boot, SPIEN
atmega328_isp_20MHz: HFUSE ?= DC
endif
# Low power xtal (20MHz) 16KCK/14CK+65ms
atmega328_isp_20MHz: LFUSE ?= FF
# 2.7V brownout
atmega328_isp_20MHz: EFUSE ?= FD
atmega328_isp_20MHz: isp
```

Dann in Kommandozeile im Verzeichnis des modifizierten Bootloaders:

```
export ENV=arduino
```

und

```
make atmega328_20MHz
```

Wichtig ist hier, dass der BAUD_RATE_CHECK erfolgreich ist:

```
avr-gcc (GCC) 5.4.0

[...]

BAUD RATE CHECK: Desired: 19200, Real: 19230, UBRRL = 129, Difference=0.1%

../../../../tools/avr/bin/avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328p
-DF_CPU=20000000L  "-DBAUD_RATE=19200 "  -DLED_START_FLASHES=3            -c -o optiboot.o
optiboot.c

../../../../tools/avr/bin/avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328p
-DF_CPU=20000000L  "-DBAUD_RATE=19200 "  -DLED_START_FLASHES=3          -Wl,--section-
start=.text=0x7e00 -Wl,--section-start=.version=0x7ffe -Wl,--relax -nostartfiles -o
optiboot_atmega328_20Mhz.elf optiboot.o

../../../../tools/avr/bin/avr-size optiboot_atmega328_20Mhz.elf

   text    data     bss     dec     hex filename

    482       0       0     482     1e2 optiboot_atmega328_20Mhz.elf

../../../../tools/avr/bin/avr-objcopy -j .text -j .data -j .version --set-section-flags
.version=alloc,load -O ihex optiboot_atmega328_20Mhz.elf optiboot_atmega328_20Mhz.hex

../../../../tools/avr/bin/avr-objdump -h -S optiboot_atmega328_20Mhz.elf >
optiboot_atmega328_20Mhz.lst

rm optiboot.o
```

Der resultierende Bootloader ist nun in Datei

```
optiboot_atmega328_20Mhz.hex
```

5.3.1.2 Arduino IDE

Als nächstes muss der Arduino IDE mitgeteilt werden, dass es die neue Board-Option gibt…

Dazu editieren von

```
arduino-1.8.9/hardware/arduino/avr/boards.txt
```

Hinzufügen

```
## sijsch
## Arduino Pro or Pro Mini (5V, 20 MHz) w/ ATmega328P
## ------------------------------------------------
pro.menu.cpu.20MHzatmega328=ATmega328P (5V, 20 MHz)

pro.menu.cpu.20MHzatmega328.upload.maximum_size=30720
pro.menu.cpu.20MHzatmega328.upload.maximum_data_size=2048
pro.menu.cpu.20MHzatmega328.upload.speed=19200

pro.menu.cpu.20MHzatmega328.bootloader.low_fuses=0xFF
pro.menu.cpu.20MHzatmega328.bootloader.high_fuses=0xDA
pro.menu.cpu.20MHzatmega328.bootloader.extended_fuses=0xFD
pro.menu.cpu.20MHzatmega328.bootloader.file=optiboot/optiboot_atmega328_20Mhz.hex

pro.menu.cpu.20MHzatmega328.build.mcu=atmega328p
pro.menu.cpu.20MHzatmega328.build.f_cpu=20000000L
```

Wichtig ist, dass die f_cpu sowie upload.speed sowie bootloader.file richtig sind.

Arduino-IDE restarten.

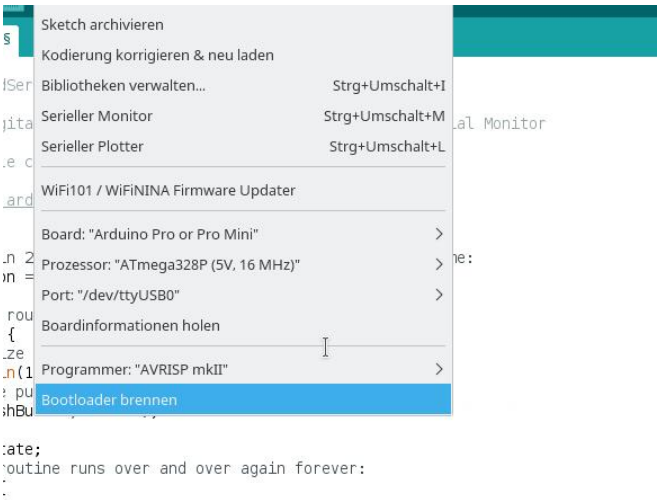
5.3.1.3 Bootloader flashen

Wir brauchen zwei Arduinos: einen als ISR-PROGRAMMER, einen als TARGET.

Auf TARGET soll der neue Bootloader.

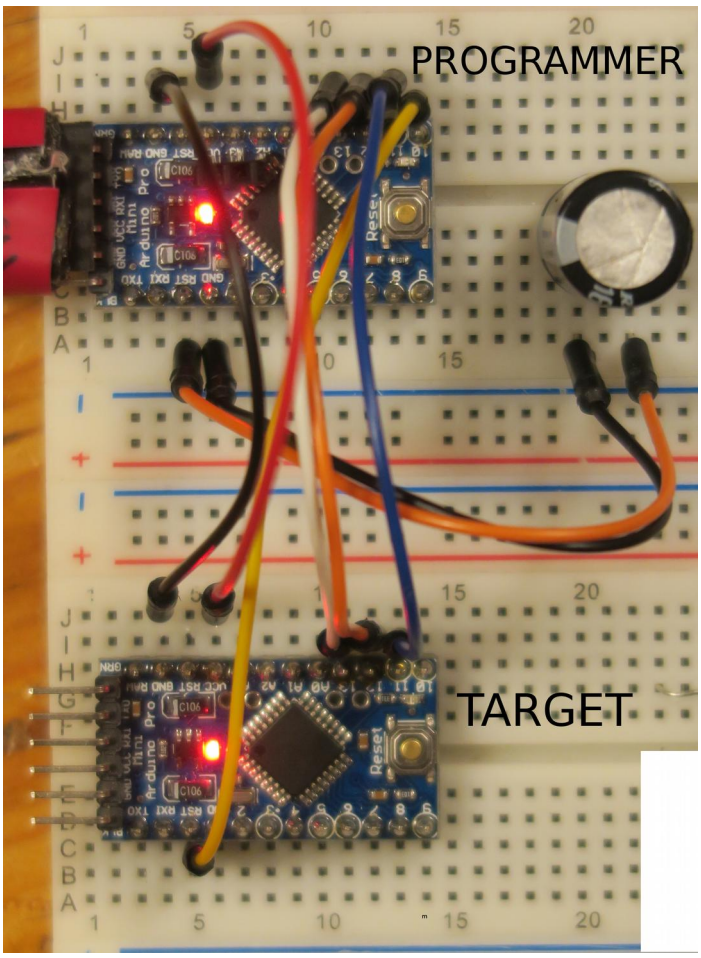
Also

1. Arduino IDE für Konfiguration von ISR-PROGRAMMER richtig einstellen: z.B.

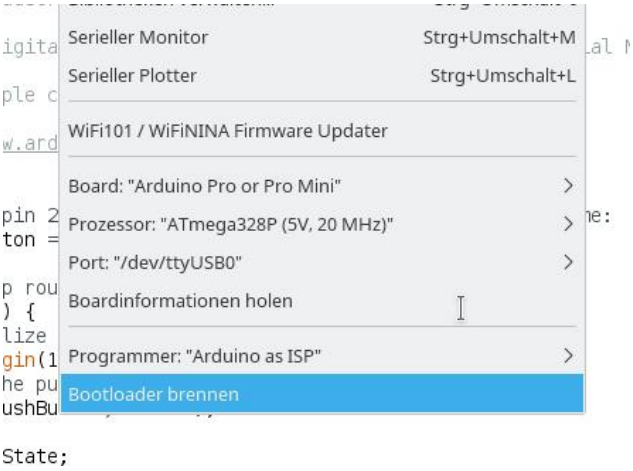


   Hier also 16Mhz und Programmer AVRISP mkII.

2. ISR-PROGRAMMER mit Sketch ArduinoISP flashen (File > Examples > ArduinoISP)
3. ISR-PROGRAMMER und TARGET verdrahten



4. Dann Arduino IDE für Konfiguration des TARGET richtig einstellen: hier also 20Mhz und Programmer Arduino as ISP



5. „Bootloader brennen“

5.3.2 20Mhz Quarz auf Arduino Pro Mini

Den Arduino Pro Mini gibt es mit 20Mhz Quarz zu kaufen (manchmal).

Wenn nur ein Standard 16Mhz Arduino Pro Mini vorhanden ist, kann dieser relativ einfach mit einem 20Mhz Quarz ausgerüstet werden.

1. Der bisherige 16Mhz Quarz muss entfernt werden

2. ein neuer 20Mhz Quarz muss aufgelötet werden.

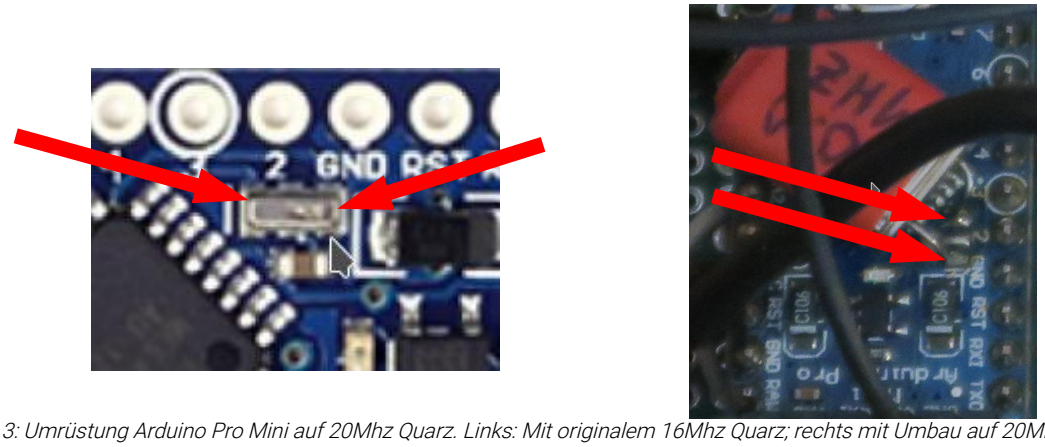


Schaubild 3: Umrüstung Arduino Pro Mini auf 20Mhz Quarz. Links: Mit originalem 16Mhz Quarz; rechts mit Umbau auf 20Mhz. Nach Entfernen des Original-Quarz sieht man drei Löt-Tabs. Die beiden äußeren sind die, an welche der neue Quarz anzuschließen ist. Der Mittlere bleibt frei.

6 Veränderungen am PAM8403-AMP-Modul

Das verwendete PAM8403-AMP-Modul muss modifiziert werden, um

1. eine kindgerechte maximale Lautstärke zu erreichen: die werkseitige maximale Lautstärke ist viel zu groß und
2. das werkseitig vorhandene Rauschen des Verstärkers auf ein akzeptables Niveau zu reduzieren.

Beide Effekte resultieren auf der werkseitig eingestellten Verstärkung von ca. 20, die viel zu groß für unsere Zwecke ist.

Aus dem Datenblatt des PAM8403 ergibt sich ein Verstärkungsfaktor von

$$A_{VD} = 20 \cdot \log\left(2 \cdot \frac{142\,k\Omega}{18\,k\Omega + R_i}\right)$$

Mit den werkseitigen 10kOhm somit

$$A_{VD} = 20 \cdot \log\left(2 \cdot \frac{142\,k\Omega}{28\,k\Omega}\right) \approx 20{,}1$$

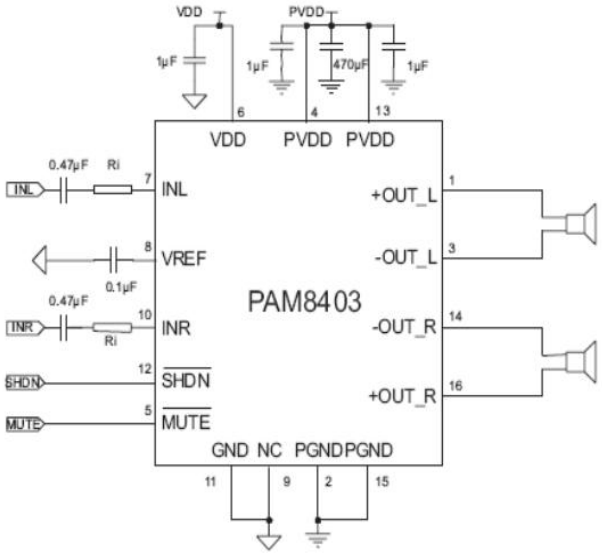


Schaubild 4: PAM8403; beim verwendeten AMP-Modul ist Ri = 10kOhm; zudem sind Pins 12 und 5 mit Vdd verbunden, wodurch eine Verwendung der Mute und Shutdown Funktionen eine Modifikation des AMP-Moduls erfordert.
[Quelle: https://www.diodes.com/assets/Datasheets/PAM8403.pdf]

Durch Ausprobieren wurde ein neuer Ri von 2MOhm als angenehm ermittelt.
Der vorhandene 10kOm Widerstand wurde durch 2MOhm ersetzt.

Zweitens wurden die MUTE und SHDN Pins mit einem Skalpell angehoben und somit von Vdd getrennt. Dies ermöglicht später eine Ansteuerung dieser Pins und damit der Mute und Suspend Funktionen des AMPs durch den Arduino.

7 Teile-Liste zum Nachkaufen

Alle hier aufgeführten Teile sind sicherlich auch in anderen Quellen erhältlich! Die Links wurden zuletzt am 29.9.2019 geprüft.

#	Was	Technische Beschreibung	z.B. Kauflink	ca. Preis
1	Arduino Pro Mini 5V 16Mhz	LINK		4€
2	(Mikro)SD Karten Leser		AMAZON	2€
3	Verstärker mit Poti und Schalter		AMAZON	2€
4	Lautsprecher		AMAZON	3€
5	Batteriehalter 4xAA ggf. mit An/Aus Schalter			3€
6	4xAA NiMH Akkus			10€
7	12x Taster mit farbigen Kappen		AMAZON	3€
8	(Mikro)SD Karte mind. Class4, z.B. 16GB			5€
9	2x Lochrasterplatine ca 7x5 cm			1€
10	Ein paar Leitungen z.B. 0,14mm²			
11	Holz tbd.			Tbd.
12	1x 74hct14 IC Schmitt-Trigger			0,5€
13	Elektro-Kleinkram (ein paar alte Kondensatoren, Widerstände, Dioden)			<2€
	Kleinkram (Schrumpfschlauch, Schrauben, ...)			

8 AudioDateien auf der SD-Karte

Wir verwenden dasselbe Format wie der Hörbert (www.hoerbert.com):

Format der Speicherkarte

Die SD-Karte muss im Format FAT32 formatiert sein. Nach Spec können FAT32 formatierte Partitionen max. 32GB groß sein. Dies entspricht mehr als 135 Stunden Musik.

Format der AudioDateien

Die Dateien müssen im PCM/WAV Format auf der SD-Karte abgelegt sein.

Sie müssen die Endung „.WAV“ haben (großgeschrieben) und in 32 KHz Samplerate, 16 Bit, Mono kodiert sein.

Für Linux wurde ein Bash-Skript geschrieben, mit dem Daten von CDs gerippt und in das richtige WAV Format gespeichert werden können.

Das Bash-Skript heißt cd2kimubo.sh.

Es nutzt die Tools

- pacpl für das Rippen von CD
- sox für die Konvertierung nach ogg und später nach WAV

Das Skript speichert die gerippten Dateien (mit hoher Qualität) als ogg und mit Kimubo-Qualität (Mono, 32kSps, 16bit) als WAV.

Die WAVs können dann auf die Kimubo-SD-Karte in das richtige Verzeichnis kopiert werden.

Ordnerstruktur und Dateien auf der Karte

Im Wurzelverzeichnis der Karte müssen Ordner mit den Namen 0 bis 8 angelegt sein!

Darin können jeweils die Audio-Dateien liegen.

Jeder Ordner entspricht einem „Werk“ (auch: Playlist). Jede Playlist kann aus 1 bis N Tracks („Liedern“) bestehen, die jeweils eine eigene Datei sind.

Beispiele:

- eine Hörspiel-CD: → „Playlist“ entspricht der CD, „Track“ entspricht den Tracks auf der CD.
- Hörspiel aus mehreren CDs: → „Playlist“ entspricht allen CDs zusammen, „Track“ entspricht den Tracks auf allen CDs

Die Playlists sind das, was mittels der bunten Tasten vom Kind ausgewählt wird.

Das Abspielen einer Playlist beginnt bei Track „0.WAV“. Automatisch werden alle Tracks einer Playlist nacheinander abgespielt (0.WAV → 1.WAV → 2.WAV → … → 9.WAV → 10.WAV → … 99.WAV → 100.WAV → …)

Es sind maximal 254 Tracks/Lieder pro Verzeichnis/Playlist möglich.

9 Software

Die Software ist in verschiedene *.ino Files aufgeteilt. Diese werden von der Arduino Entwicklungsumgebung automatisch beim Compilieren zu einer großen *.c Datei zusammengeführt[5]:

- kimubo.h
- kimubo.ino
- kimubo_010_globals.ino
- kimubo_020_pcmplayer.ino
- kimubo_030_amp.ino
- kimubo_040_sdfet.ino
- kimubo_050_volpot.ino
- kimubo_060_ubat.ino
- kimubo_070_keypad.ino
- kimubo_080_sleeper.ino
- kimubo_090_messages.ino
- kimubo_900_setup-function.ino
- kimubo_950_loop-function.ino

9.1 Libraries

Es werden folgende Libs verwendet:

- avr/power.h
- EEPROM.h
- SdFat.h
- LKpcm.h
- Keypad.h und Key.h

Besondere Bedeutung hat die Lkpcm-Lib, s.u.

9.2 Programm-Aufbau

Generell wird viel mit globalen Variablen und wenig mit OOP gearbeitet.

Alle vom User konfigurierbaren Variablen sind in kimubo.h definiert.

Alle sonstigen globalen Variablen und Funktions-Definitionen sind in kimubo_010_globals.ino definiert.

kimubo_070_keypad.ino enthält die Tastatur-Abstraktion. Dort wird auf Key-Press, Key-Hold, Key-Release Events reagiert, indem Funktionen aufgerufen werden, z.B. Funktionen des PCM-Players. Die Tastuatur ist vollständig abstrahiert, d.h. Key-Events enthalten selbst keine Funktionslogik, sondern rufen immer nur „fremde“ Funktionen auf.

kimubo_900_setup-function.ino initialisiert das System und enthält die Arduino-setup().

kimubo_950_loop-function.ino enthält die Arduino-loop().

5 https://github.com/arduino/Arduino/wiki/Build-Process

Es wird auf neue Key-Events geprüft.
Es wird geprüft, ob der Loudness-Jumper gesetzt ist.
Es wird geprüft, ob der PCM-Player (der Interupt-gesteuret wird) das Track-Ende meldet – und ggf. wird der Player mit dem Spielen des nächsten Tracks beauftragt.
Es wird beim ersten Durchlaufen der Loop() geprüft, ob automatisch mit dem zuletzt gespielten Track weitergemacht wird (Autoplay).

kimubo_020_pcmplayer.ino enthält die Betriebslogik des PCM Players. Die Abspielfunktion selbst ist in die Lkpcm Library ausgelagert.

kimubo_040_sdfet.ino enthält die Prüfung der SDKarte.

9.3 Lkpcm-Library

Enthält die Abspielfunktion für WAV/PCM Dateien von SD-Karte.

Im Wesentlichen handelt es sich um eine Interupt-Service-Routine (ISR), die passend zur Anzahl der Samples aufgerufen wird: bei 32kSps also 32000 mal pro Sekunde. In dieser werden die jeweils aktuellen WAV-Werte aus einem Puffer entnommen und an den Timer/Counter1 des Arduino übergeben. Somit stellt dieser die entsprechende PWM-Ausgangssignale an Pins 9 und 10, die dann (s.o.) zu einem Analog-Signal kombiniert und verstärkt werden.

Der Puffer wird von einer zweiten ISR befüllt. Diese nimmt einen Block von Daten von der SD-Karte und stellt diese in den Puffer.

Die erste ISR kann alles andere (auch die zweite ISR) unterbrechen, sie ist real-time-kritisch.

10 Funktionale Details

10.1 Funktionale Details zu SD-Karte

Damit der Arduino die SD-Karte lesen kann, wird die SDFAT Library verwendet. Diese spart gegenüber der Standard-Arduino SD-Library Speicherplatz und ist ein bisschen performanter. Detail dazu siehe https://github.com/greiman/SdFat.

10.2 Funktionale Details zu Audio-Ausgabe (PCM/WAV Dekodierung)

Siehe oben.

10.3 Funktionale Details zu Auswahl des Werkes, Anspringen der Stücke in einem Werk und Vor-/Zurückspulen

Die Werke sind das, was mittels der bunten Tasten vom Kind ausgewählt wird.

Innerhalb eines Werks kann mittels Kurzdruck der grauen „Lied vor/Lied zurück“ Tasten je ein Lied vorgesprungen oder zurückgesprungen werden (Skipt-Funktion).

10.4 Funktionale Details zu ISPC Anschluss

Der normale Arduino Pro Mini ISPC Anschluss bleibt erhalten und dient dazu, den Arduino auch im System neu zu programmieren bzw. zu debuggen.

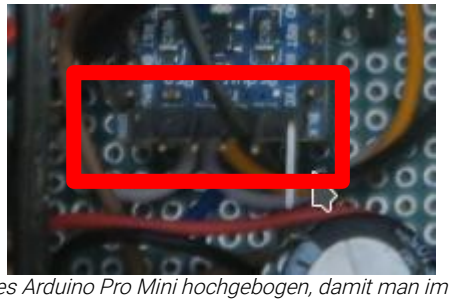


Abbildung 1: ISPC Pins; an der Schmalseite des Arduino Pro Mini hochgebogen, damit man im System besser drankommt

11 Gehäuse

Abmessungen
20 x 14 x 8,5

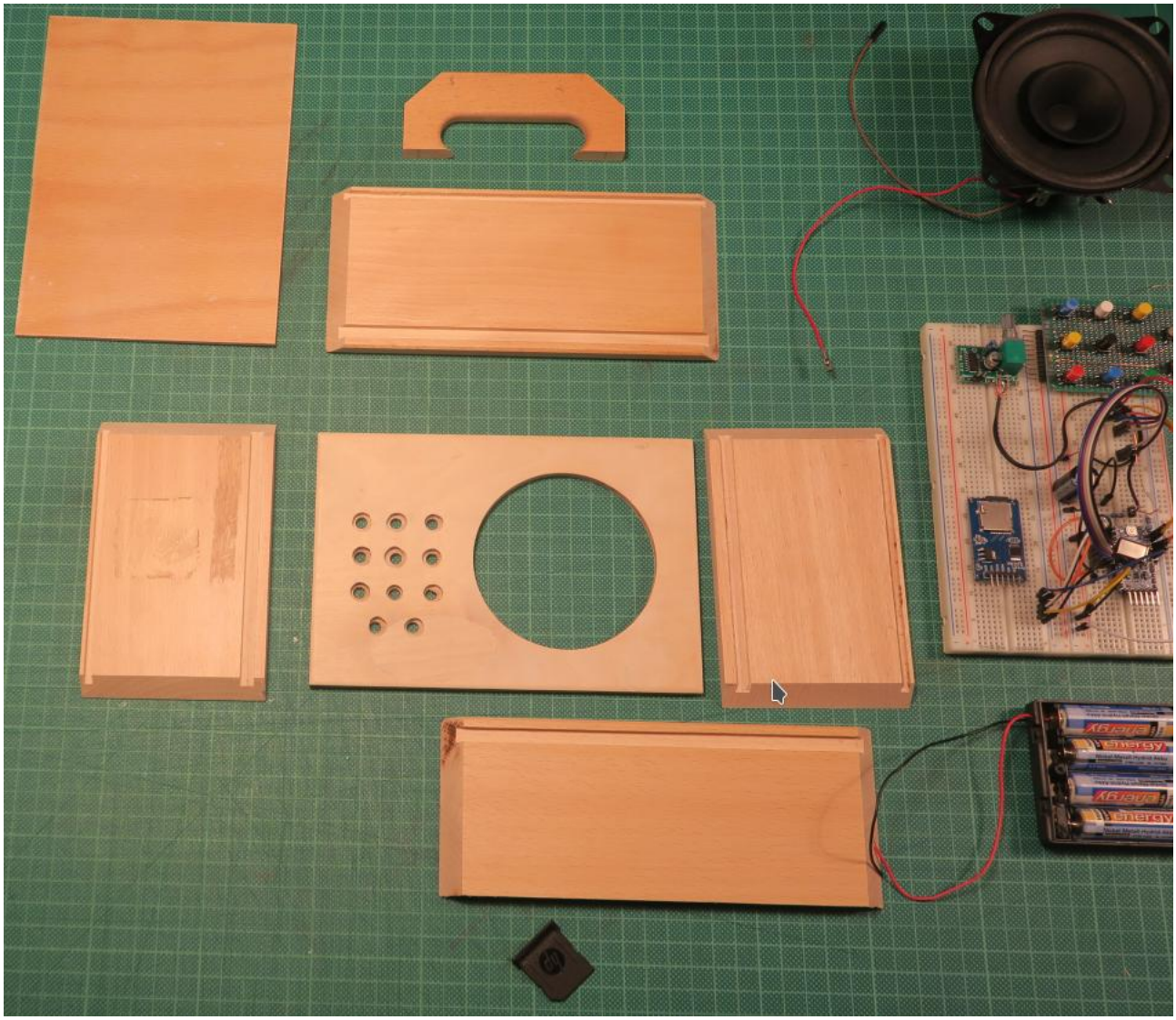


Abbildung 2: Gehäuse und Innereinen eines Kimubo

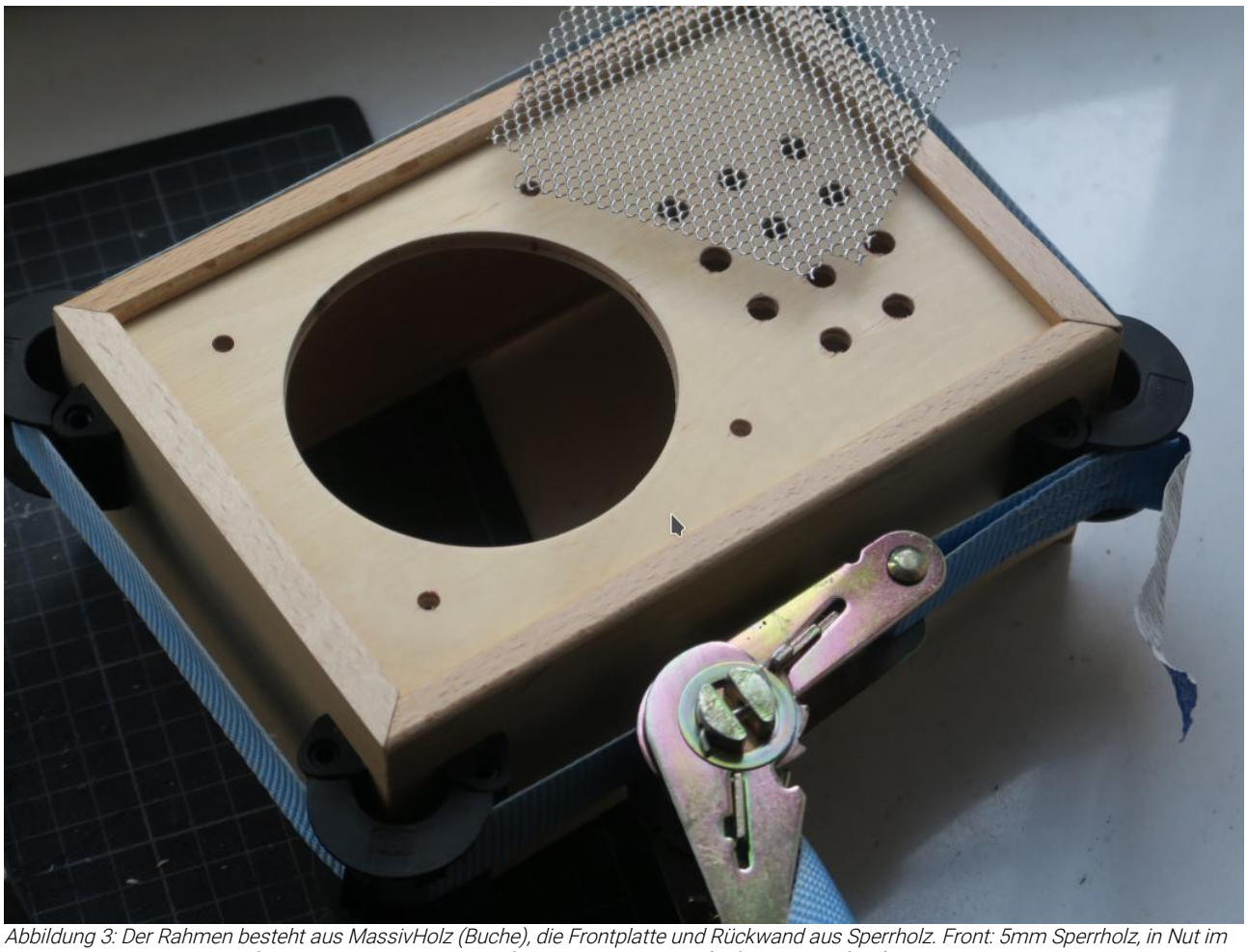


Abbildung 3: Der Rahmen besteht aus MassivHolz (Buche), die Frontplatte und Rückwand aus Sperrholz. Front: 5mm Sperrholz, in Nut im Rahmenholz eingeleimt. Auf der Rückseite hinter Tastenfeld in Dicke reduziert (gefräste Vertiefung). Lautsprecher wird geschraubt. Rückplatte: 4mm Buchen-Sperrholz. Eingeschoben in Nut im Rahmenholz. Rahmen auf Gehrung verleimt.

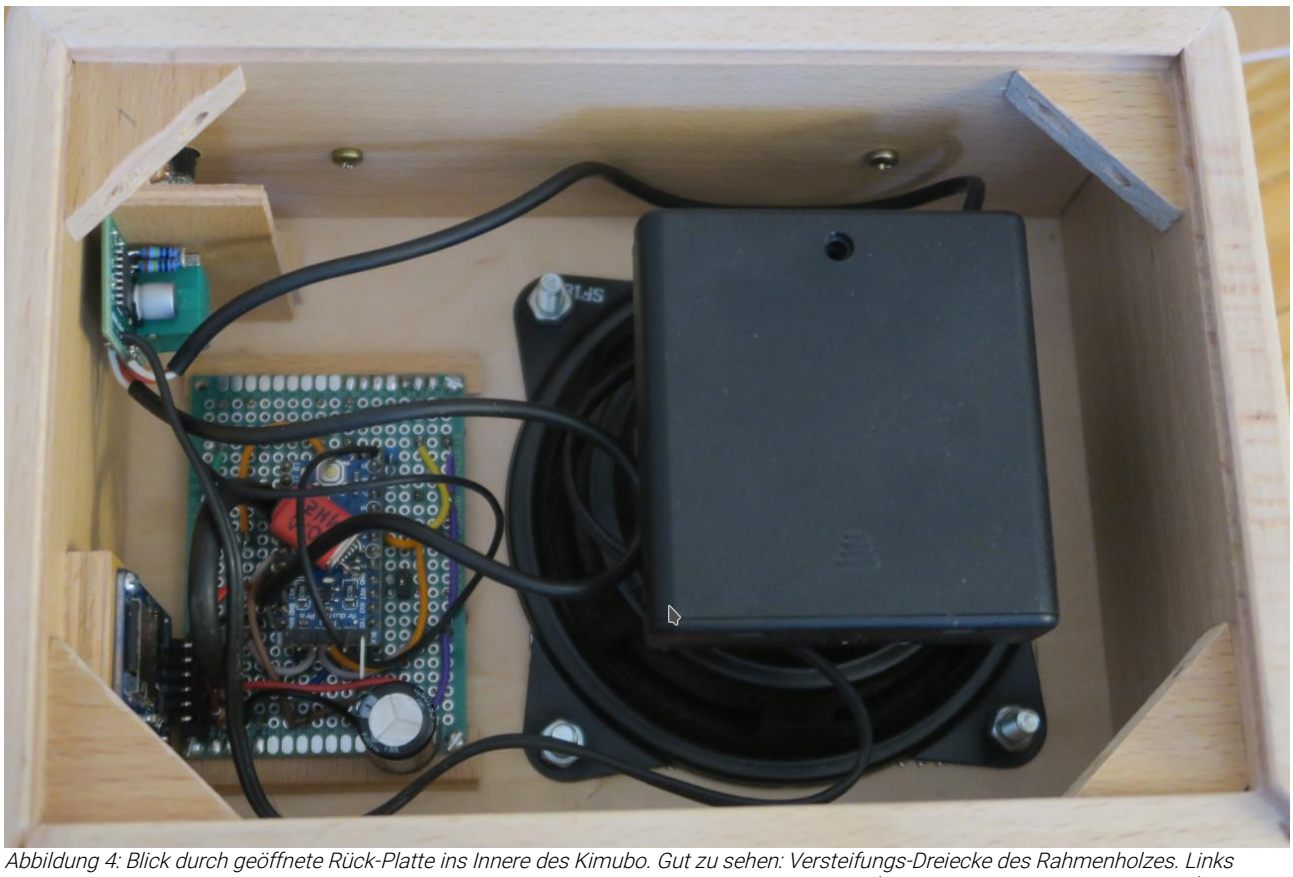


Abbildung 4: Blick durch geöffnete Rück-Platte ins Innere des Kimubo. Gut zu sehen: Versteifungs-Dreiecke des Rahmenholzes. Links unten der senkrechte SD-Karte-Leser mit Micro-SDC. Daneben Lochraster-Platine mit Tastern (von hier aus: Rückseite der Platine) und dem Arduino (aufgesteckt) und Puffer-Elko. Links oben das Verstärker-Modul mit Poti; befestigt an Poti an einer zurückgesetzten Halte-Wand aus Spreeholz im Innern des Geräts. Rechts Lautsprecher mit am MAgneten aufgeklebter 4xAA-Batterie-Halterung.

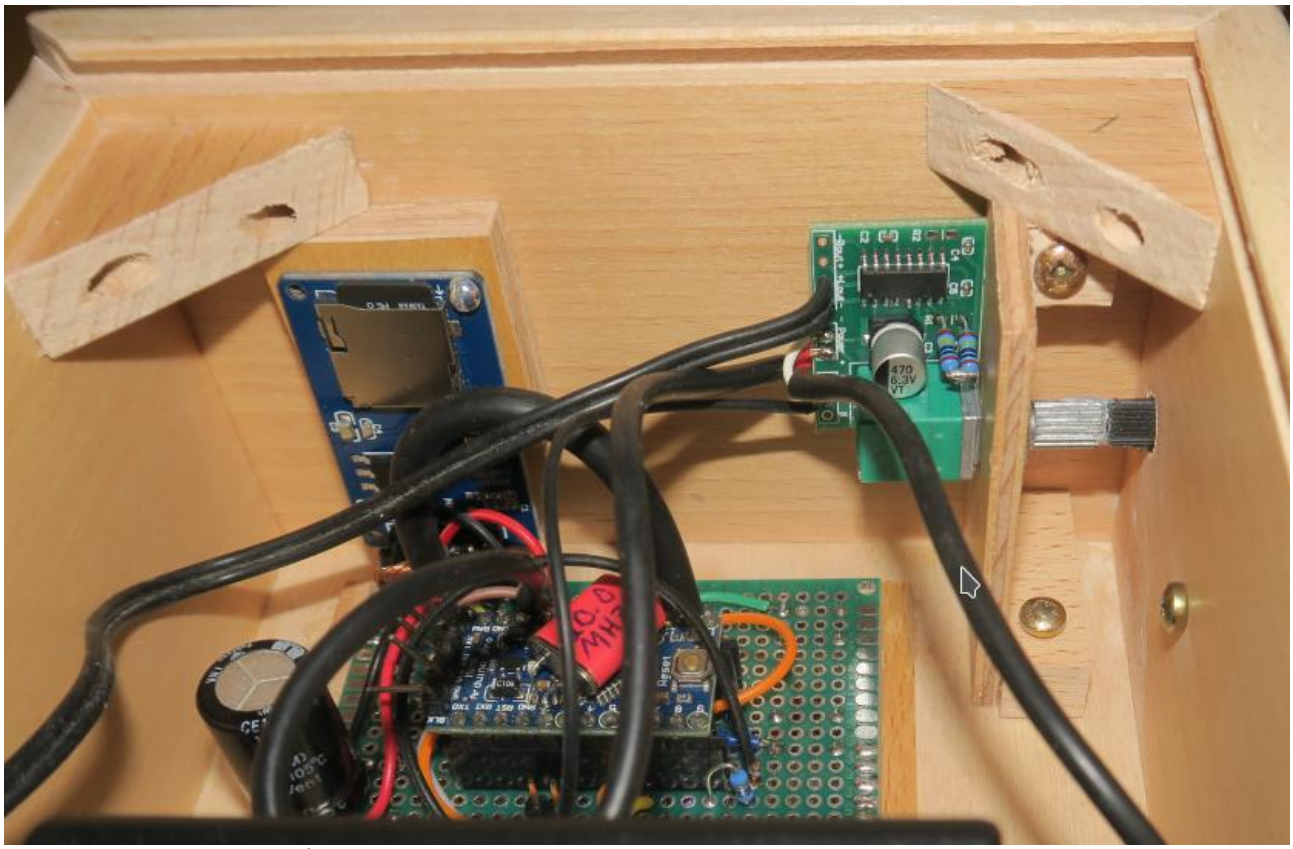


Abbildung 5: Noch eine Sicht auf die Elektronik

12 In Action