

Neue Ideen für alte Handys und Smartphones:

Elektronische Basteleien mit DTMF

Von Miguel Köhnlein und Michael Gaus



- DTMF-Decodermodule mit MT8870 günstig erhältlich
- Drahtlos steuern per Bluetooth durch Verwendung eines Bluetooth-Headset-Adapters
- Kabelgebunden steuern per Kopfhörerbuchse (z.B. mit 1:1 Kabel mit Klinkensteckern)
- Alternativ kann Headset gehackt/umgebaut werden, wenn keine Klinkenbuchse vorhanden
- Funktioniert auch mit älteren Handys (also keine Smartphones), wenn die Telefontastatur DTMF-Töne erzeugt
- Am Smartphone wird nicht unbedingt eine App benötigt, Telefontastatur reicht
- Alternativ sind Apps mit DTMF-Ausgabe für Android und iPhone verfügbar
- DTMF-Sequenzen können auch als MP3-Files mit dem Musikplayer ausgegeben werden
- Arduino-Anbindung des DTMF-Moduls über 5 Eingangspins (Taktsignal und 4 Bit Daten)
- Lichteffekte können passend zur Musik gesteuert werden durch Einbettung von DTMF-Tönen in die Musikdatei

Kurzüberblick

Als wir in der Make 2/18 den Aufruf „Neue Ideen für alte Handys“ gelesen haben, war unsere Überlegung, ob es eine Möglichkeit gibt, Handys möglichst universell als Fernsteuerung benutzen zu können. Möglichst unabhängig davon, ob älteres Handy oder Smartphone, ob Android oder iPhone, ob aktuelle OS-Version oder ältere, ob bluetooth-fähig oder nicht, idealerweise ohne spezielle App. Fast jedes Handy bietet die Möglichkeit, Audiosignale auszugeben, entweder kabelgebunden per Kopfhörer- oder Headsetausgang oder drahtlos per Bluetooth-Audio-Interface. Zudem generieren fast alle Modelle beim Betätigen einer Taste auf der Telefontastatur einen DTMF-Ton, der dann auch über die Audio-Schnittstelle ausgegeben werden kann. Viele Handys haben auch einen Musikplayer, sodass auch DTMF-Töne durch Abspielen einer entsprechenden Musikdatei erzeugt werden können.

Somit entstand dieses Projekt, um mit DTMF-Signalen elektronische Bastelprojekte (z.B. mit Arduino) per Handy über die Telefontastatur oder den Musikplayer steuern zu können. Alternativ gibt es für Android und iPhone auch Apps, die eine DTMF-Tastatur bereitstellen. Als zu steuernde Hardware wird ein Arduino Uno mit dem Multifunction-Shield verwendet, auf der Siebensegmentanzeige können die empfangenen DTMF-Codes angezeigt werden. Es wird sowohl eine kabelgebundene Anbindung ans Handy als auch eine drahtlose Steuerung per Bluetooth mittels eines Bluetooth-Headset-Adapters gezeigt. Abschließend wird noch beschrieben, wie sich das Handy als Musikplayer mit Lichteffektsteuerung passend zur Musik einsetzen lässt, beispielsweise um den heimischen Partykeller aufzumotzen. Im Beispiel wird ein Arduino mit einem WS2812 RGB LED-Stripe verwendet, der per DTMF-Signal an bestimmten Stellen in einer Musikdatei Steuerbefehle erhält, um entsprechende Lichtmuster zu aktivieren. Zur Veranschaulichung ist ein Video hierzu verfügbar.

Inhaltsverzeichnis

1.	DTMF-Signale.....	2
2.	DTMF-Decoder	3
3.	Anbindung an Arduino	4
4.	DTMF-Signale per Handy senden	7
5.	Drahtlose Steuerung mit Bluetooth.....	9
6.	Lichteffektsteuerung per DTMF passend zur Musik.....	12

1. DTMF-Signale

DTMF ist die Abkürzung für „dual-tone multi-frequency“, übersetzt bedeutet das so viel wie „Doppelton-Mehr frequenz“. Es können bis zu 16 Töne durch Überlagerung von jeweils zwei sinusförmigen Signalen erzeugt werden. Dieses Verfahren ist auch unter dem Namen Mehrfrequenzwahlverfahren bekannt, das bei der analogen Telefontechnik zur Übermittlung der gewünschten Rufnummer an die Vermittlungsstelle verwendet wird. Wenn man auf dem Handy eine Telefonnummer wählt, werden dort fast immer auch DTMF-Töne auf dem Lautsprecher ausgegeben (teilweise kann dies in den Einstellungen konfiguriert werden). Schließt man ein Headset an, so werden die Töne dort ausgegeben und können mit einem geeigneten Decoder ausgewertet werden.

Zuordnung der DTMF-Frequenzpaare zu den Tasten:

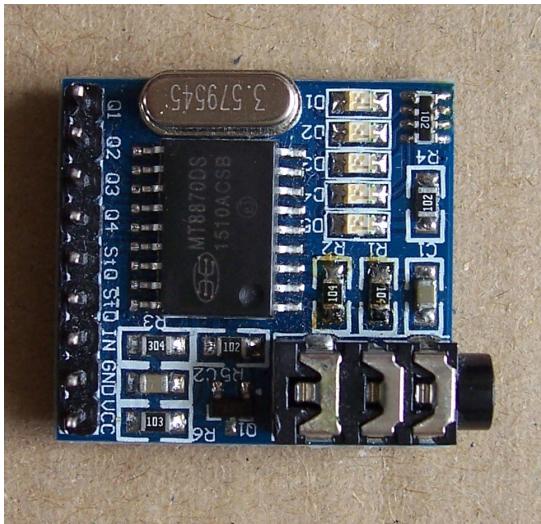
	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

2. DTMF-Decoder

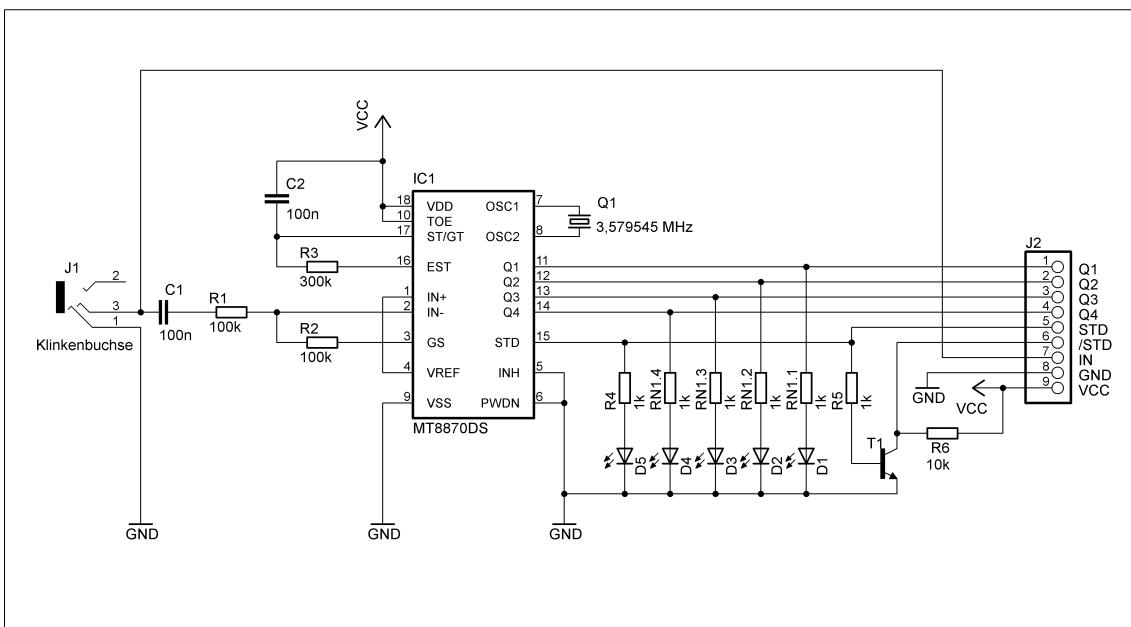
Passende DTMF-Decodermodule mit dem MT8870 gibt es fertig aufgebaut günstig zu kaufen, beispielsweise hier:

<https://www.ebay.de/itm/MT8870-DTMF-Sprach-Audio-Decoder-Telefon-Modul-fur-Arduino-Phone/171557999296>

Foto 1: DTMF Decodermodul



Die Schaltung des DTMF-Moduls mit MT8870 wurde per Durchgangsprüfer ausgemessen, der Schaltplan sieht folgendermaßen aus:



Das DTMF-Audiosignal wird über eine Klinkenbuchse eingespeist, wobei nur der rechte Kanal der Stereobuchse angeschlossen ist. Das Signal wird zunächst intern über einen Operationsverstärker im MT8870 verstärkt. Über das Widerstandsverhältnis R2/R1 ist die Verstärkung des invertierenden Verstärkers einstellbar. Falls das Eingangssignal zu gering sein sollte, kann der Widerstand R1 verkleinert werden. Der Eingangspegel muss laut Datenblatt im Bereich zwischen 21 und 869 mV liegen. Damit ein DTMF-Ton erkannt wird, muss dieser außerdem eine Mindestdauer von ca. 50 ms aufweisen. Zwischen den Tönen muss eine Mindestpause im ungefähr gleichen Zeitbereich vorhanden sein.

Das dekodierte Signal wird über die Pins Q1 bis Q4 bereitgestellt. Zusätzlich wird der 4-bit Code über die LEDs D1 bis D4 angezeigt. Das Bitmuster bleibt solange stehen, bis ein neues DTMF-Signal erkannt wurde. Während der Dauer des Tonsignals wird der Ausgang STD auf HIGH-Pegel gezogen, was auch durch Aufleuchten der LED D5 sichtbar ist. Dadurch kann man STD als Taktsignal zur flankengesteuerten Datenübernahme verwenden. Zusätzlich wird dieses Signal auch invertiert über einen Transistor am Pin /STD bereitgestellt.

Die DTMF-Signale erzeugen folgende Datenmuster an Q1 bis Q4:

Taste auf Telefon	Code an Q4...Q1 (binär)	Code (hex)
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
0	1010	A
*	1011	B
#	1100	C
A	1101	D
B	1110	E
C	1111	F
D	0000	0

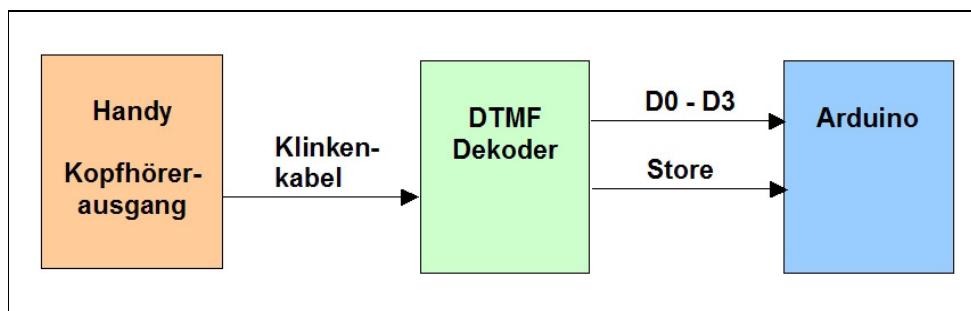
Die Tasten A bis D sind auf der Telefontastatur normalerweise nicht vorhanden. Bei manchen Apps mit DTMF-Tonerzeugung sind diese jedoch verfügbar.

3. Anbindung an Arduino

Die Anbindung des DTMF-Moduls an ein Handy oder Smartphone kann im einfachsten Fall direkt über den Kopfhörerausgang erfolgen. Bei einer vorhandenen 3,5 mm Kopfhörerbuchse wird nur ein 1:1 Verbindungskabel jeweils mit Klinkenstecker benötigt. Manche Handys haben eine 2,5 mm Klinkenbuchse, hier hilft dann ein passendes Adapterkabel. Falls keine Kopfhörerbuchse vorhanden ist, besteht oft die Möglichkeit, über eine spezielle Buchse ein passendes Headset anzuschließen. Durch Hacken bzw. Umbauen eines solchen Headsets kann das Audiosignal dann abgegriffen werden. Meist muss nur der Ohrhörer abgeschnitten und stattdessen ein Klinkenstecker angelötet werden.

An die Ausgänge Q1 bis Q4 sowie STD des DTMF-Moduls kann ein Arduino angeschlossen werden. Hierzu werden 5 Eingangspins am Arduino benötigt. Idealerweise schließt man die 4 Datenleitungen am gleichen Port des AVR-Mikrocontrollers und in passender Reihenfolge (Q1 ist LSB) an, sodass softwareseitig ohne großen Aufwand das DTMF-Signal weiterverarbeitet werden kann.

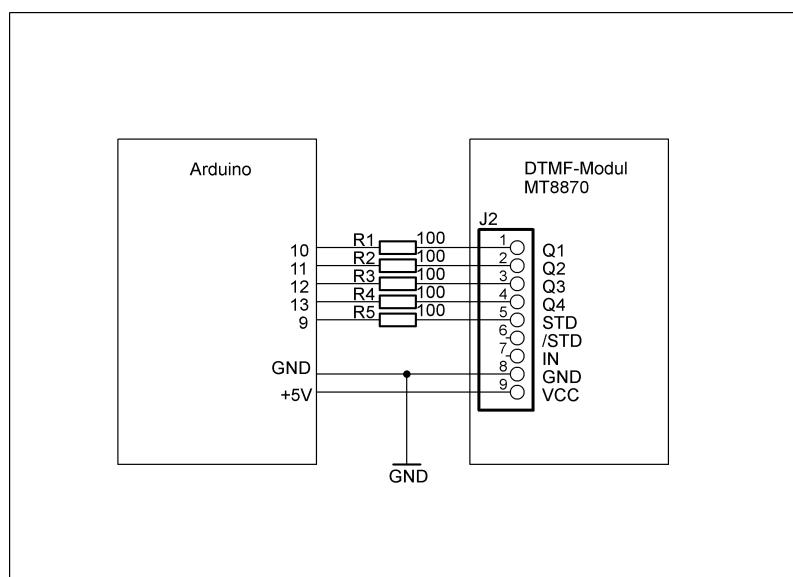
Das Blockschaltbild zur Anbindung eines Arduinos an das Handy sieht folgendermaßen aus:



Im Beispiel hier wurde ein Arduino Uno mit dem Multifunction Shield verwendet. Als Eingangspins für die Daten wurden die Anschlüsse 10 bis 13 (entspricht PB2 bis PB5) verwendet, wobei eine Stifteleiste an die Lötpins des entsprechenden Arduino-Steckers auf dem Shield angelötet wurde. **Achtung:** auf dem Shield ist an diesen 4 Pins jeweils eine LED angeschlossen, d.h. bei den meisten Anwendungen werden diese Pins vermutlich als Ausgänge konfiguriert sein. In der Anwendung hier zur DTMF-Auswertung müssen diese Pins jedoch als Eingänge konfiguriert werden. Man sollte generell immer vor dem Aufspielen eines anderen Arduino-Sketchs das Shield abziehen (bei ausgeschalteter Versorgungsspannung), dann den anderen Sketch aufspielen und erst anschließend wieder das Shield aufstecken (ebenfalls bei ausgeschalteter Versorgungsspannung). Nur so ist gewährleistet, dass es zu keinem Konflikt mit der angeschlossenen Hardware kommen kann und womöglich zwei Ausgänge gegeneinander treiben. Zur Sicherheit wurden hier deshalb die Widerstände R1 bis R5 vorgesehen, sodass im Fall des Falles der Strom begrenzt wird. Wenn man sich sicher ist, dass die Arduino-Pins als Eingänge beschaltet sind, können R1 bis R5 auch weggelassen werden.

Das Taktsignal STD für die Datenübernahme wurde an Pin 9 des Arduinos angeschlossen, dieser ist auf dem MFS-Shield an einer Stifteleiste zusammen mit der Versorgungsspannung +5V und GND verfügbar.

Schaltplan:



Arduino Sketch

Als Vorlage wurde das Beispiel „mfs_timer1“ verwendet und entsprechend angepasst.

Die Zeile

const uint8_t LED1 = 13;
wurde entfernt, da die LEDs hier nicht verwendet und die Pins stattdessen als Eingänge beschaltet werden.

Über den Define `PIN_DTMF_STD` wird der Arduino-Pin des STD-Datenübernahmesignals definiert. Mit `PINS_DTMF_DATA` wird der Eingangsport für die Daten festgelegt, wobei hier die Bezeichnung PINx des entsprechenden AVR-Ports benötigt wird. `PINS_DTMF_DATA_LSB` gibt die Bitnummer des Pins für das LSB (also Q1) an. Q2 bis Q4 müssen dann in aufsteigender Reihenfolge bezogen auf diesen Pin angeschlossen sein.

Pin 10 des Arduinos entspricht PINB.2 des ATmega328. Passend zum Schaltplan sehen die Defines folgendermaßen aus:

```
#define PIN_DTMF_STD 9  
#define PINS_DTMF_DATA PINB  
#define PINS_DTMF_DATA ISR 2
```

In der setup-Routine wurden die Aufrufe für LED1 entfernt. Stattdessen wird der Pin STD als Eingang definiert, wobei dies nicht unbedingt erforderlich wäre, da nach einem Reset alle Ports zunächst Eingänge sind. Deshalb wurde auf das explizite Konfigurieren der 4 Datenpins als Eingänge verzichtet.

Die setup-Routine sieht folgendermaßen aus:

```
void setup()
{
    pinMode(LATCH, OUTPUT);
    pinMode(CLK, OUTPUT);
    pinMode(DATA, OUTPUT);
    pinMode(BUZZER, OUTPUT);
    pinMode(BUTTON1, INPUT);      // I/O-Pin als Eingang
    digitalWrite(BUZZER, HIGH);   // BUZZER aus (Low-Aktiv)

    pinMode(PIN_DTMF_STD, INPUT);

    TCCR1A = 0;                  // Register loeschen
    OCR1A = 1000;                // Vergleichswert x = (CPU / (2 x Teiler x f)) - 1
    TCCR1B |= (1 << CS10) | (1 << CS11) | (1 << WGM12); // CTC-Mode, Teiler = 64
    TIMSK1 |= (1 << OCIE1A);   // Output Compare A Match Interrupt Enable
    sei();                      // IRQ Enable
}
```

Die loop-Routine wurde komplett ersetzt und sieht nun folgendermaßen aus:

```
void loop()
{
    uint8_t dtmfValue, dtmfState;
    static uint8_t dtmfState_old = LOW;
    uint16_t value;

    dtmfState = digitalRead(PIN_DTMF_STD);
    if( (dtmfState == HIGH) && (dtmfState_old == LOW) )
    {
        dtmfValue = PINS_DTMF_DATA;
        dtmfValue = dtmfValue >> PINS_DTMF_DATA_LSB;
        if(dtmfValue >= 10)
        {
            dtmfValue = 0;
        }
        value = DisplayValue;
        value = ((uint32_t)value * 10) % 10000;
        value += dtmfValue;
        DisplayValue = value;
    }
    dtmfState_old = dtmfState;
}
```

Bei jedem Durchlauf wird der STD-Pin eingelesen. Wird ein Flankenwechsel erkannt, also aktueller Pin-Zustand HIGH und vorheriger Zustand in der Variablen dtmfState_old LOW, dann werden die Datenpins eingelesen und ausgewertet. Auf der 7-Segmentanzeige wird dann die gedrückte Taste als Ziffer 0 bis 9 ganz rechts dargestellt, wobei die zuvor gedrückten Tasten um eine Stelle nach links geschoben werden. Es sind also immer die zuletzt gedrückten vier Tasten auf der Anzeige sichtbar. Falls eine Taste außerhalb des Ziffernbereichs (z.B. * und #) gedrückt wurde, wird diese als Null dargestellt.

Anstatt den Pin STD zu pollen ist auch eine Auswertung per Interrupt möglich. Hierzu müsste dann ein Pinchange-Interrupt konfiguriert werden und in der Interruptserviceroutine auf die steigende Flanke geprüft werden.

Somit kann mit diesem Beispiel ein einfacher Test der Steuermöglichkeiten per DTMF-Signal erfolgen und die gedrückten Zifferntasten auf der Anzeige dargestellt werden. Je nach Bedarf können die empfangenen Signale individuell ausgewertet werden, um beispielsweise eine Fernbedienung per Handy aufzubauen.

4. DTMF-Signale per Handy senden

Um auf dem Handy die DTMF-Signale zu erzeugen, genügt im einfachsten Fall bereits die Telefontastatur. Je nach Modell muss evtl. die DTMF-Ausgabe (Tastentöne) aktiviert und ggf. die Lautstärke erhöht werden. Durch Anschluss eines Kopfhörers oder Headsets kann dann zunächst geprüft werden, ob bei einem Tastendruck ein Tonsignal am Audio-Ausgang erzeugt wird. Nach Anschluss des DTMF-Moduls am Handy kann man an den 4 LEDs auf dem DTMF-Modul direkt erkennen, ob ein DTMF-Signal detektiert wurde. Falls das Ausgangssignals des Handys zu schwach sein sollte, dann kann evtl. auch die Verstärkung auf dem DTMF-Modul durch Umbestückung eines Widerstands erhöht werden, wie weiter vorne bereits beschrieben.

Foto 2: Steuerung per Kopfhörerbuchse, ohne spezielle App



Foto 3: Sogar mit einem solch altertümlichen Handy ist eine Steuerung möglich



Als Alternative zur Telefontastatur können auch DTMF-Töne in Form einer Musikdatei ausgegeben werden, falls das Handy über einen Musikplayer verfügt. Die DTMF-Einzeltöne der Tasten 0 bis 9 sowie Stern und Raute stehen auf der folgenden Seite in einer zip-Datei als MP3-Files zur Verfügung:
<https://www.mediacollege.com/audio/tone/dtmf.html>

Mit einem Audio-Bearbeitungstool wie z.B. Audacity können mehrere Töne hintereinandergefügten und dann durch Abspielen der entsprechenden Musikdatei auf dem Handy eine DTMF-Sequenz verschickt werden. Es gibt auch einen Online-Tongenerator, um DTMF Sequenzen als WAV-Datei zu erzeugen:
https://www.audiocheck.net/audiocheck_dtmf.php

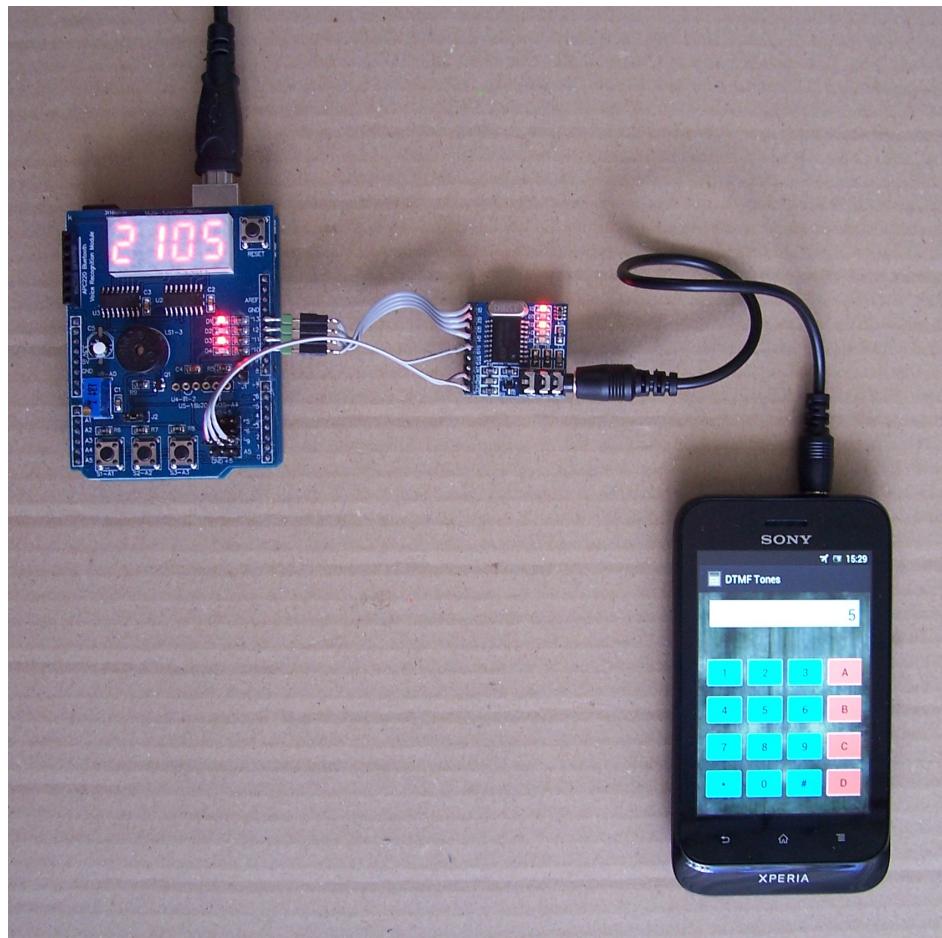
Falls ein Smartphone verwendet wird, gibt es auch Apps, um DTMF-Signale per Tastendruck zu erzeugen.

Android-Apps:
„DTMF Tones“
https://play.google.com/store/apps/details?id=com.thechampanurag.dtmftonegenerator&hl=en_US

„Simply DTMF“
<https://play.google.com/store/apps/details?id=net.simplyadvanced.simplytonegenerator>

iPhone-App:
„Dialer“
<https://itunes.apple.com/us/app/dialer/id482793462?mt=8>

Foto 4: Steuerung per Kopfhörerbuchse, mit der App „DTMF Tones“



5. Drahtlose Steuerung mit Bluetooth

Anstatt der kabelgebundenen Variante ist auch eine drahtlose Steuerung per Bluetooth möglich. Hierzu wird zusätzlich noch ein Bluetooth-Headset-Adapter (Receiver) mit Kopfhörerausgang benötigt.

Foto 5: Bluetooth-Headset-Adapter mit Kopfhörerausgang



Das hier verwendete Modell hat einen integrierten Akku:

<https://www.pearl.de/a-HZ2037-1050.shtml?query=Headset+Adapter+mit+Bluetooth+4+1>

Alternativ sollte auch ein Modell diesen Typs funktionieren:

<https://www.ebay.de/itm/Lapel-3-5mm-Sport-Bluetooth-4-1-Receiver-Headset-Adapter-Audio-Transmitter-G6/263586824439>

Es gibt auch Varianten ohne eingebautem Akku, die per USB mit Strom versorgt werden:

<https://www.ebay.de/itm/Bluetooth-Audio-Receiver-Auto-KFZ-Adapter-AUX-klinke-3-5mm-Kabel-USB-Empfanger/253376842074>

Die Bedientasten dieser Headsets werden nicht benötigt. Das Smartphone wird per Bluetooth mit dem Headset gekoppelt. Die DTMF-Tastentöne werden dann per Bluetooth an das Headset gesendet und von dort per Kopfhörerausgang an das DTMF-Decodermodul weitergegeben.

Dieser Mechanismus funktioniert auch auf dem iPhone, im Gegensatz zu Bluetooth SPP (Serial Port profile), das dort softwaremäßig blockiert ist, sodass die günstigen Bluetooth-SPP-Uart-Module nicht funktionieren. Da bei der DTMF-Anwendung jedoch nur Bluetooth-Audioprofile für Headset-Steuerung benötigt werden, kann auch per iPhone ein Arduino gesteuert werden.

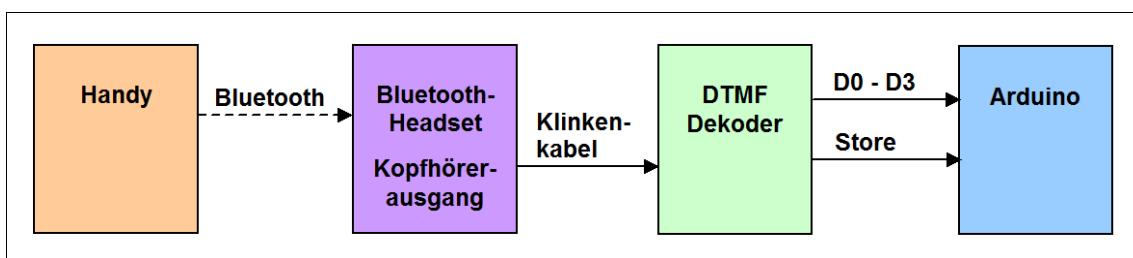
Foto 6: Drahtlose Steuerung per Bluetooth-Headset-Adapter



Foto 7: Drahtlose Steuerung per Bluetooth-Headset-Adapter, mit der App „DTMF Tones“



Blockschaltbild für Steuerung per Bluetooth:

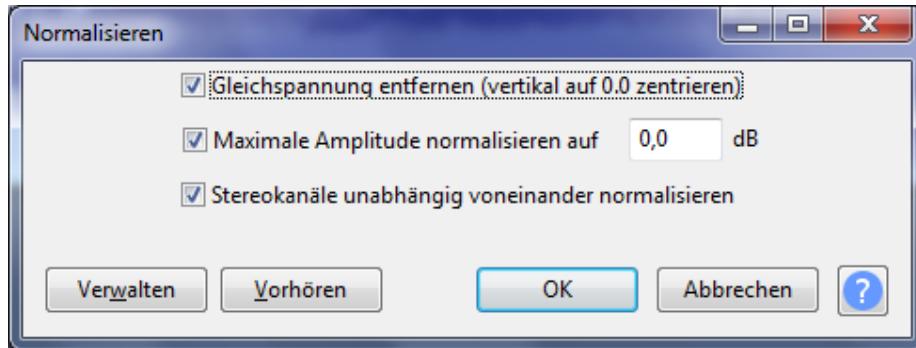


6. Lichteffektsteuerung per DTMF passend zur Musik

Um Lichteffekte passend zur Musik generieren zu können, fügen wir an den entsprechenden Stellen in der Musikdatei DTMF-Töne ein. Damit diese nicht in der Musik hörbar sind, wird die Musik auf dem linken Kanal und die DTMF-Töne auf dem rechten Kanal ausgegeben. Über entsprechende Audio-Adapterkabel wird dann der linke Kanal auf beide Eingangskanäle des angeschlossenen Aktiv-Lautsprechers oder Verstärkers und der rechte Kanal auf das DTMF-Decodermodul geschaltet. Um die Musikdatei editieren zu können, eignet sich das Freeware-Programm Audacity:

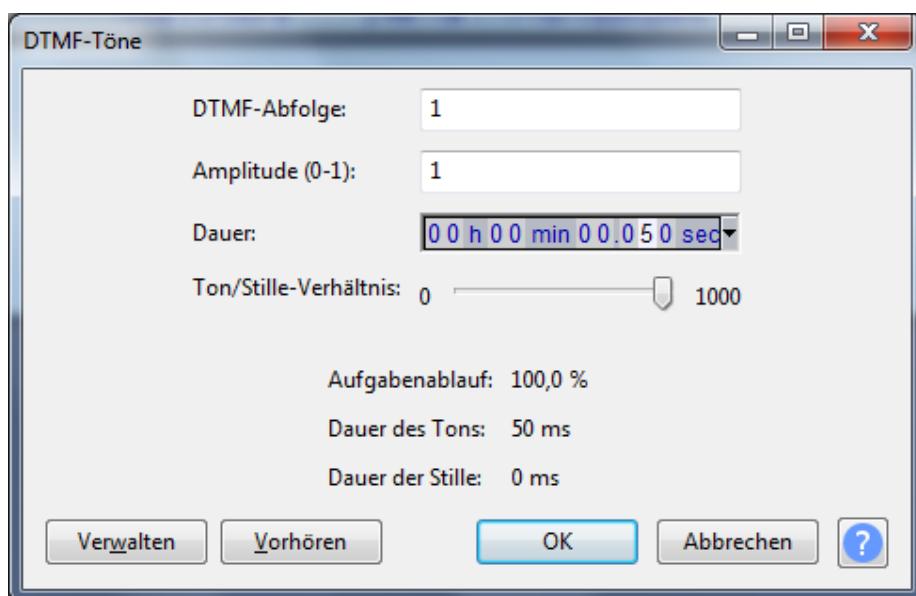
<https://www.audacityteam.org/>

Zunächst öffnen wir die Musikdatei, die üblicherweise im Stereo-Format vorliegt, mit Audacity. Es werden zwei Bereiche für die beiden Musikkanäle (links und rechts) mit der Wellenform des Musiksignals angezeigt. Anschließend wird der komplette Bereich mit Strg-A markiert und im Menü bei „Effekt“ die Funktion „Normalisieren“ ausgewählt. Dadurch wird erreicht, dass später alle Musikdateien ungefähr gleiche Lautstärke haben.

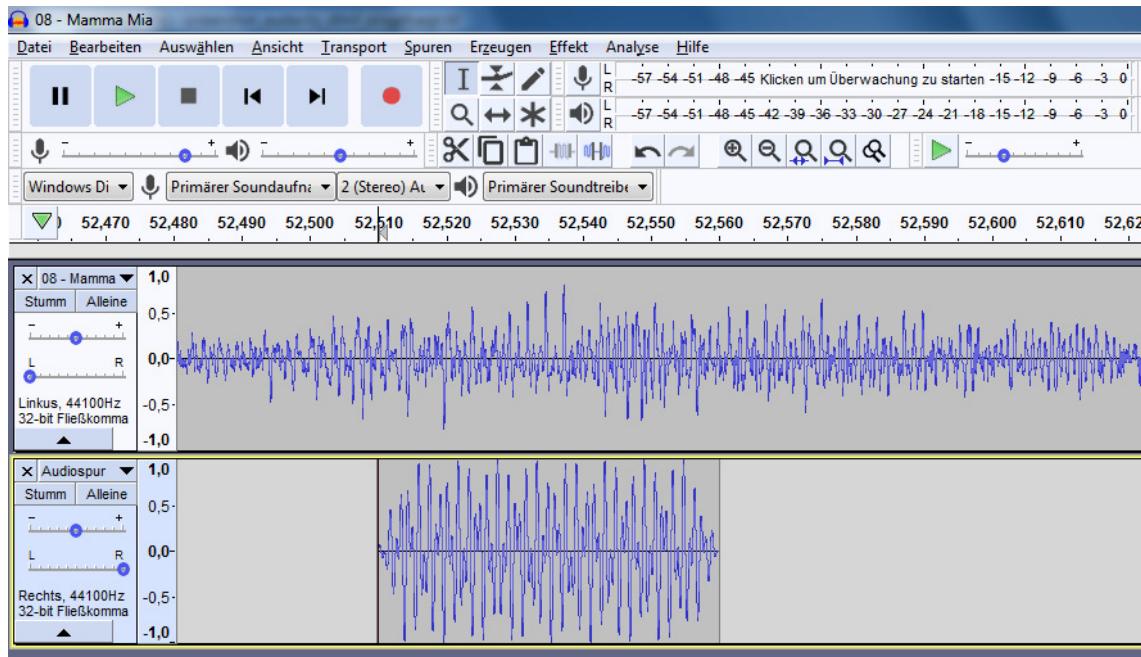


Danach wird unter „Spuren“ => „Mix“ => „Stereo zu Mono heruntermischen“ ausgewählt. Nun sollte statt der zwei Bereiche nur noch ein Bereich mit der Wellenform dargestellt werden. Über „Spuren“ => „Neu hinzufügen“ => „Monospur“ erzeugen wir einen zweiten leeren Bereich. Im oberen Bereich mit der Wellenform stellen wir den Kanal-Schieberegler ganz nach links auf die Position „L“. Im unteren leeren Bereich wird der Kanal-Schieberegler ganz nach rechts auf die Position „R“ geschoben.

Nun können wir im unteren leeren Bereich an den gewünschten Stellen DTMF-Töne einfügen. Hierzu kann per Cursor an die gewünschten Stellen gesprungen und mit Play/Pause vorgehört werden. Mit der Lupe kann hineingezoomt werden. Hat man die passende Stelle gefunden, wird mit der Maus in den unteren leeren Bereich hineingeklickt, um den Cursor zu positionieren. Über „Erzeugen“ => „DTMF-Töne“ erscheint folgender Dialog:



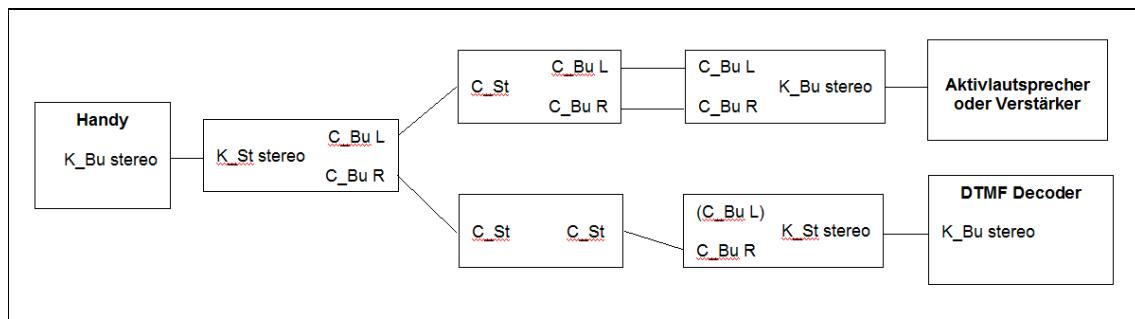
Bei DTMF-Abfolge wird die gewünschte Taste eingetragen, möglich sind 0 bis 9, Stern *, Raute #, sowie die 4 Zusatztasten A bis D (Großschreibung). Die Amplitude stellen wir auf 1, die Dauer auf 50 ms, entsprechend 00h 00min 00,050sec. Durch Klick auf OK wird der Ton eingefügt und sollte nun als Wellenform mit einer Dauer von 50ms im unteren Bereich sichtbar sein.



Nachdem alle gewünschten DTMF-Töne eingefügt wurden, sollte zunächst die Projektdatei mit „Datei“ => „Projekt speichern unter“ gespeichert werden, damit später evtl. weitere Änderungen durchgeführt werden können. Anschließend wird die Musikdatei erzeugt mit „Datei“ => „Exportieren“ und dann z.B. im MP3-Format gespeichert.

Es sind somit 16 verschiedene DTMF-Steuerbefehle in der Musikdatei möglich. Je nachdem wie man später die DTMF-Signale auswertet, könnte man auch jeweils 2 Töne kurz hintereinander erzeugen, um die die Anzahl Befehle zu erhöhen. Ebenso denkbar wäre es, beim Start eines Lieds eine mehrstellige Nummer per DTMF zu übertragen, um das Lied zu kennzeichnen und dann anschließend immer denselben DTMF-Ton an den entsprechenden Stellen einzusetzen, um nur den nächsten Lichteffekt für die Steuerung passend zur Liednummer zu markieren. Im Beispiel hier verwenden wir jedoch die 16 Befehlsmöglichkeiten.

Um die passende Audio-Verkabelung herzustellen muss nicht unbedingt gelötet werden, es reicht eine Adapterlösung mit Klinken- und Cinch-Kabeln.



K_Bu = Klinkenbuchse, K_St = Klinkenstecker, C_Bu = Cinch-Buchse, C_St = Cinch-Stecker

Den Adapter Cinch-Stecker auf 2x Cinch-Buchse gibt es beispielsweise hier:
<https://www.pollin.de/p/cinchadapter-450374>

Durch die beschriebene Verkabelung erhält der Stereo-Aktiv-Lautsprecher bzw. Stereo-Verstärker das Mono-Musiksignal (linker Kanal der Handy-Audioausgabe) auf beiden Eingängen rechts und links. Das Mono-DTMF-Signal (rechter Kanal der Handy-Audioausgabe) wird auf den DTMF-Decoder geschaltet, dort ist beim beschriebenen Modul nur der rechte Eingangskanal der Klinkenbuchse beschaltet. Möchte man eine drahtlose Steuerung per Bluetooth-Audio realisieren, dann wird der Klinkenstecker in den Bluetooth-Headset-Adapter eingesteckt statt in die Handy-Kopfhörerbuchse.

Wenn man nun auf dem Handy die mit DTMF-Signalen präparierte Musikdatei abspielt, sollte auf dem Aktivlautsprecher die Musik zu hören sein und an den entsprechend markierten Stellen ein DTMF-Befehl beim Decodermodul ankommen. Dies kann man daran erkennen, dass dort die Leuchtdiode D5 kurz aufblitzt und an D1 bis D4 das entsprechende Bitmuster des DTMF-Befehls angezeigt wird. Mit dem passenden Arduino-Sketch kann dann ein WS2812 RGB LED-Stripe mit den DTMF-Befehlen gesteuert werden, sodass zur Musik passende Lichteffekte erzeugt werden.

Da in diesem Fall gilt: „ein Video sagt mehr als 1000 Bilder“, gibt es statt eines Fotos ein kleines Filmchen mit der Musik-Lichteffektsteuerung per Bluetooth über das Handy in Aktion:
<https://youtu.be/bTn-upDpmIk>

Arduino-Sketch für Lichteffektsteuerung per DTMF

Der Sketch basiert auf dem weiter vorne verwendeten DTMF-Sketch.

Am Anfang wird folgendes für die Neopixel-Bibliothek eingefügt, wobei der passende LED-Stripe angegeben werden muss:

```
#include <Adafruit_NeoPixel.h>

#define PIXEL_PIN      5      // Digital IO pin connected to the NeoPixels.
#define PIXEL_COUNT   60

// Parameter 1 = number of pixels in strip, neopixel stick has 8
// Parameter 2 = pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_RGB      Pixels are wired for RGB bitstream
//   NEO_GRB      Pixels are wired for GRB bitstream, correct for neopixel stick
//   NEO_KHZ400   400 KHz bitstream (e.g. FLORA pixels)
//   NEO_KHZ800   800 KHz bitstream (e.g. High Density LED strip), correct for
neopixel stick
Adafruit_NeoPixel strip = Adafruit_NeoPixel(PIXEL_COUNT, PIXEL_PIN, NEO_GRB +
NEO_KHZ800);
```

In der Setup-Routine wird die Initialisierung für den LED-Stripe hinzugefügt:

```
strip.begin();
strip.setBrightness(64);
strip.show(); // Initialize all pixels to 'off'
```

Die loop-Routine sieht folgendermaßen aus:

```
void loop()
{
    uint8_t dtmfValue, dtmfState;
    static uint8_t dtmfState_old = LOW;
    uint16_t value;

    dtmfState = digitalRead(PIN_DTMF_STD);
    if( (dtmfState == HIGH) && (dtmfState_old == LOW) )
    {
        dtmfValue = PINS_DTMF_DATA;
        dtmfValue = dtmfValue >> PINS_DTMF_DATA_LSB;

        switch(dtmfValue)
        {
            case 0:
```

```

for (uint16_t i = 0; i < strip.numPixels(); i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0));
}
strip.show();
break;

case 1:
for (uint16_t i = 0; i < strip.numPixels(); i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0));
}
for (uint16_t i = 0; i < 3; i++)
{
    strip.setPixelColor(i, strip.Color(0xFF, 0, 0));
}
strip.show();
break;

case 2:
for (uint16_t i = 0; i < strip.numPixels(); i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0));
}
for (uint16_t i = 3; i < 6; i++)
{
    strip.setPixelColor(i, strip.Color(0, 0xFF, 0));
}
strip.show();
break;

case 4:
for (uint16_t i = 0; i < strip.numPixels(); i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0));
}
for (uint16_t i = 6; i < 9; i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0xFF));
}
strip.show();
break;

case 8:
for (uint16_t i = 0; i < strip.numPixels(); i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0));
}
for (uint16_t i = 9; i < 12; i++)
{
    strip.setPixelColor(i, strip.Color(0xFF, 0, 0xFF));
}
strip.show();
break;

case 0xF:
for (uint16_t i = 0; i < strip.numPixels(); i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0));
}
for (uint16_t i = 0; i < 12; i++)
{
    strip.setPixelColor(i, strip.Color(0xFF, 0xFF, 0xFF));
}
strip.show();
break;

case 3:
for (uint16_t i = 0; i < strip.numPixels(); i++)
{
    strip.setPixelColor(i, strip.Color(0, 0, 0));
}

```

```
    for (uint16_t i = 0; i < 6; i++)
    {
        strip.setPixelColor(i, strip.Color(0xFF, 0xFF, 0));
    }
    strip.show();
    break;

    case 0xC:
    for (uint16_t i = 0; i < strip.numPixels(); i++)
    {
        strip.setPixelColor(i, strip.Color(0, 0, 0));
    }
    for (uint16_t i = 6; i < 12; i++)
    {
        strip.setPixelColor(i, strip.Color(0xFF, 0xFF, 0));
    }
    strip.show();
    break;
}

if(dtmfValue >= 10)
{
    dtmfValue = 0;
}
value = DisplayValue;
value = ((uint32_t)value * 10) % 10000;
value += dtmfValue;
DisplayValue = value;
}
dtmfState_old = dtmfState;
}
```