2 Sound und Timer

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

2 Sound und Timer 2
schnelles Takten für Anfänger 2
Ziele 2
Video 2
LCD Display ansteuern - wie kommt Test auf eine Anzeige? 2
Ziele 2
Video 2
Übung 2
To Do: notwendige header-Dateien 7

2 Sound und Timer

Details dazu sind auch unter Die Timer und Zähler des AVR bei mikrocontroller.net zu finden.

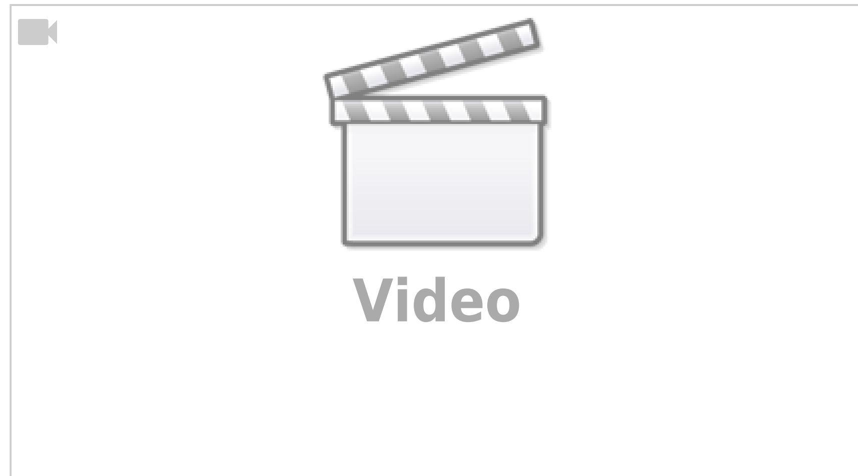
schnelles Takten für Anfänger

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie man im Atmel Studio eine Puls-Signal ausgibt

Video



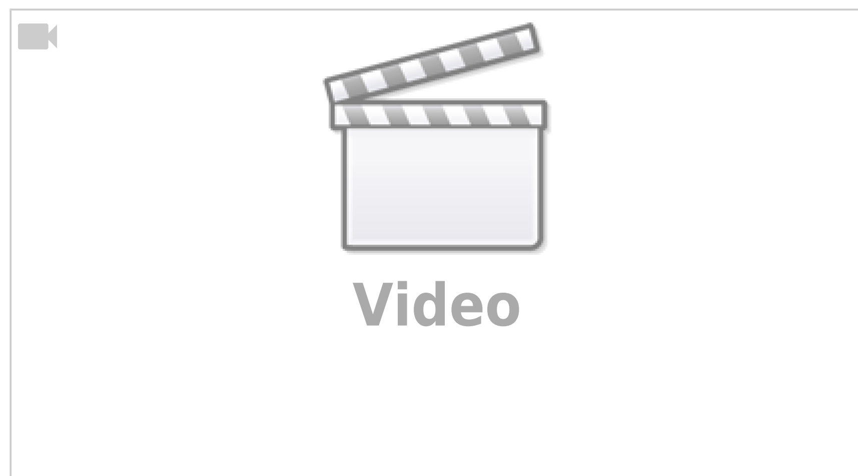
LCD Display ansteuern - wie kommt Test auf eine Anzeige?

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie Display angesprochen wird

Video

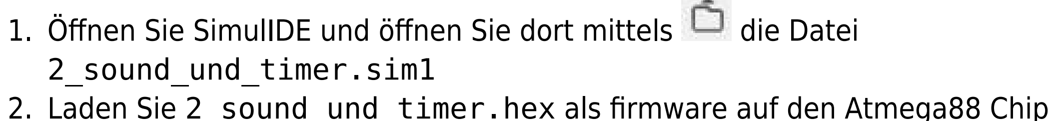


Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 2._sound_und_timer.sim1
    2. 2._sound_und_timer.hex
    3. lcd_lib_de.h

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE und öffnen Sie dort mittels  die Datei `2_sound_und_timer.sim1`
    2. Laden Sie `2_sound_und_timer.hex` als firmware auf den Atmega88 Chip
2. Betrachtung der neuen Komponenten
    1. In der Simulation sind nun neben der LED weitere Komponenten zu sehen, die im Folgenden erklärt werden sollen
        1. ein Display Hd44780
        2. zwei Frequenzmesser, mit der Anzeige 0Hz
        3. ein Lautsprecher mit Schalter
        4. ein Oszilloskop
    2. Starten Sie die Simulation
        1. Wenn Sie die Lautsprecher des PCs aktiv haben, werden Sie ein Knacken (und ggf. Brummen) vernehmen. Dies rührt von der unvollständigen Simulation her. Wenn Sie den den Schalter vor dem Lautsprecher schließen, so sollten Sie - zusätzlich zum Knacken - einen aufsteigenden und abfallenden Ton hören. Es ist möglich, dass dieser nach wenigen Sekunden aufhört, auch das ist eine Eigenschaft der unvollständigen Simulation.
        2. Am Frequenzmesser ist dennoch zu sehen, dass ein Signal mit Frequenzen zwischen $500...1000~\rm Hz$ ausgegeben wird.
        3. Das Oszilloskopbild zeigt den Signal-Zeit-Verlauf, in welchem die länger und kürzer werdenden Wechsel von HIGH nach LOW sichtbar sind.
        4. Die LED zeigt nun an, ob es sich um einen aufsteigenden oder abfallenden Ton handelt
        5. Das Display zeigt zusätzlich an, welches Experiment geladen ist
    3. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Eingabe in Atmel Studio

1. öffnen Sie Atmel Studio
2. legen Sie ein neues “GCC C Executable Project” mit dem Namen `2_sound_und_timer` für einen ATmega88 an
3. Bevor das eigentliche Coding beginnt sollte immer eine Beschreibung dem Code vorangestellt werden. Hierzu kann folgende Vorlage verwendet werden:

```
/*=================================================================
=============

Experiment 2:   Sound-Generator
=============   ===============

Dateiname:      2_sound_und_timer.c

Autoren     :    Peter Blinzinger
                Prof. G. Gruhler     (Hochschule Heilbronn,
Fakultät T1)
                D. Chilachava       (Georgische Technische
Universität)
```

```
                T. Fischer       (Hochschule Heilbronn, Fakultät
T1)

Version:        1.3 vom 19.09.2022

Hardware:       MEXLE2020 Ver. 1.0 oder höher
                AVR-USB-PROGI Ver. 2.0

Software:       Entwicklungsumgebung: AtmelStudio 7.0
                C-Compiler: AVR/GNU C Compiler 5.4.0

Funktion:       Auf einem kleinen Lautsprecher (Buzzer) wird nur
einem MEXLE-Board
                ein sirenenartiger Sound ausgegeben. Zwischen den
auf-
                und absteigenden Tönen bleibt die Frequenz kurz
stabil.
                Die Frequenz wird mit dem Timer 0 (im CTC-Mode)
erzeugt und
                direkt über den Output-Compare-Pin im Toggle-Mode
ausgegeben.

Displayanzeige: +----------------+
                |- Experiment 2 -|
                | Creating Sound |
                +----------------+

Tastenfunktion: keine

Jumperstellung: Schalter muss fuer den Buzzer betätigt sein

Fuses im uC:    CKDIV8: Aus        (keine generelle Vorteilung des
Takts)

Header-Files:   lcd_lib_de.h    (Library zur Ansteuerung LCD-
Display Ver. 1.3)

==================================================================
===========*/
```

Ändern Sie die folgenden Informationen, je nach Programm:

1. `Autor`: Der folgende Code basiert auf eine Version verschiedener Autoren, diese sind hier angegeben. Wenn Sie einen eigenen Code generieren sollte hier Ihr Name stehen. Dies ermöglicht eine Nachvollziehbarkeit bei Unklarheiten
2. `Version`: Um die Aktualität der Software zu erkennen sollte mindestens das Datum angegeben und bei Änderung immer aktualisiert werden
3. `Hardware`: Die getestete bzw. verwendete Hardware sollte angegeben werden, um nachvollziehen zu können, wie der Code getestet werden kann. Bei Simulationen sollte hier mindestens der verwendete Chipsatz (z.B. ATmega88) angegeben werden.

4. `Software`: Zum weiterverwenden des Codes ist die Angabe der Entwicklungsumgebung (engl. integated Development environment IDE) wichtig. Nicht selten gibt es bei größeren Projekten Schwierigkeiten, wenn IDE und Compiler geändert werden.
5. `Funktion`: Die Funktion des Programms sollte kurz erklärt werden. Damit wird dem Leser bereits vor dem Code schon Hinweise gegeben
6. `Display-Anzeige`: Ähnlich der Funktion ist auch eine Darstellung der (erwartbaren) Anzeige sinnvoll.
7. `Tastenfunktion`: Bei zukünftigen Anwendungen kann eine Eingabe von Tastenstellungen sinnvoll sein. Dies sollte hier angegeben werden
8. `Jumperstellungen`: Jumper bieten die Möglichkeit unterschiedliche Schaltungsteile der Hardware zu verbinden oder zu trennen. Damit können Hardwarefunktionalitäten aktiviert oder deaktiviert werden. Wenn verschiedene Jumperstellungen für ein Programm wichtig sind, so sollten diese angegeben werden.
9. `Fuses im uC`: Beim Microcontroller (auch µC oder uC abgekürzt) bieten die Möglichkeit interne Konfigurationen anzupassen. Diese werden über Fuses eingestellt werden. Sind hier Funktionen für das Programm notwendig, so sollten diese angegeben werden
10. `Header-Files`: Werden weitere Softwareteile genutzt, so sollten diese über Header-Dateien eingebunden. Diese sollten bereits schon vor dem eigentlichen Codeteil kurz erklärt werden.

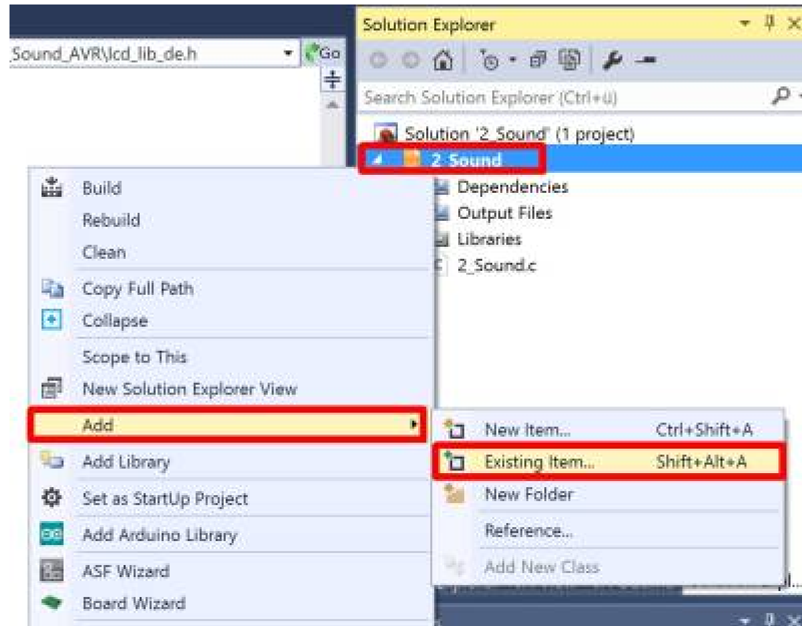
4. Nach der Beschreibung steht im Code der Deklarationsbereich:

```
// Deklarationen
==============================================================

// Festlegung der Quarzfrequenz
#ifndef F_CPU                   // optional definieren
#define F_CPU 18432000UL        // MiniMEXLE nutzt einen ATmega88
mit 18,432 MHz Quarz
#endif

// Include von Header-Dateien
#include <avr/io.h>               // I/O Konfiguration (intern
weitere Dateien)
#include <util/delay.h>           // Definition von Delays
(Wartezeiten)
#include "lcd_lib_de.h"           // Funktionsbibliothek zum LCD-
Display

// Konstanten
#define MIN_PER     143          // minimale Periodendauer in
"Timerticks"
#define MAX_PER     239         // maximale Periodendauer in
"Timerticks"
#define WAIT_TIME   2000        // Wartezeit zwischen Flanken in
ms

// Makros
#define SET_BIT(BYTE, BIT)    ((BYTE) |=  (1 << (BIT))) // Bit
```

```
Zustand in Byte setzen
#define CLR_BIT(BYTE, BIT)  ((BYTE) &= ~(1 << (BIT))) // Bit
Zustand in Byte loeschen
#define TGL_BIT(BYTE, BIT)  ((BYTE) ^=  (1 << (BIT))) // Bit
Zustand in Byte wechseln (toggle)

// Funktionsprototypen
void initDisplay(void);         // Initialisierung Display und
Startanzeige
void initPorts(void);           // Initialisierung der I/O-Ports
void initTimer(void);           // Timer 0 initialisieren
(Soundgenerierung)
void init(void);                // generelle
Initialisierungsfunktion
```

Bei den Deklarationen werden Vorgaben gemacht, welche wichtig sind, bevor der Code vor dem eigentlichen Prozessor oder Controller ausgeführt werden. Die Deklarationen weisen den Präprozessor an, bestimmte Vorgaben zu nutzen (Details zu Präprozessor und Compiler-Direktiven sind hier zu finden). Folgende Punkte sollten mindestens angegeben werden:

1. `Quarzfrequenz`: Die Taktfrequenz des Microcontrollers kann entweder intern oder extern definiert werden. Diese Frequenz sollte immer angegeben werden. Wird dies nicht vorgenommen, kann es Probleme bei der Handhabung von Wartezeiten ("Delays") geben. In Simulide kann die Frequenz des externen Quarz eingegeben werden - diese sollte zum in der Software angegebenen Frequenz passen. Die Angabe `#ifndef` ist hier eine Compiler-Direktive und keine C-Code. Wie alle anderen Deklarationen sind alle Zeilen nach der einem # vor der Ausführung des Codes im Controller zur Zeit der hexfile-Erstellung im Compiler wichtig. `#ifndef` prüft hierbei, ob ein Symbol bereits in einem anderen File definiert wurde.
2. `Header includes`: Header-Dateien sollten bereits aus der Informatik 2 bekannt sein. Bei IDEs wird häufig zwischen Dateien unterschieden, welche in den Ordnern der IDE und Dateien, welche im Projektordner liegen. Hier sollen folgende Header-Dateien genutzt werden:
    1. `<avr/io.h>`: Header-Datei, welche Input/Output Bezeichner für Pins und Ports definiert. Da im Folgenden bestimmte Ports angesprochen werden sollen, ist diese Header-Datei wichtig. Die Header-Datei liegt im Unterordner `avr` in den Ordnern der IDE. Diese wurde bereits schon im Bespiel 1_hello_blinking_world verwendet.
    2. `<util/delay.h>`: Header-Datei, welche einen einfachen Umgang mit Wartezeiten ermöglicht. Diese wurde bereits schon im Bespiel 1_hello_blinking_world verwendet.
    3. `"lcd_lib_de.h"`: Diese Header-Datei sollte im Projektordner eingefügt sein. Bei einem neuen Projekt ist sie dies noch nicht.



To Do: notwendige header-Dateien

Bitte fügen Sie die Datei lcd_lib_de.h in den Projektordner ein
Dazu sollten Sie im Solution Explorer auf das Projekt rechts-klicken (hier `2. Sound und Timer`) » Add » Existing Item… (siehe Bild rechts).
Die gewünschte Datei (hier: die heruntergeladene `lcd_lib_de.h`) auswählen und mit Add hinzufügen. Die Datei sollte nun im Solution Explorer angezeigt werden.

3. `#defines`: Über `#defines` kann vorgegeben werden, welcher Text durch den Präprozessor im Code ersetzt werden soll. Damit können Konstanten oder kurze Codeersetzungen (Makros) vorgegeben werden. In diesem Programm soll ein maximale und minimale Periodendauer, sowie eine Haltedauer für den höchsten und niedrigsten Ton vorgegeben werden. Zusätzlich sind drei Standardmakros vorgegeben, um
   1. ein Bit in einem Byte zu setzen (`SET_BIT(BYTE, BIT)`),
   2. ein Bit in einem Byte zu löschen (`CLR_BIT(BYTE, BIT)`), oder
   3. ein Bit in einem Byte zu invertieren (`TGL_BIT(BYTE, BIT)`).
4. `Funktionsprototypen`: Wie regulär bei der C-Programmierung sollten die verwendeten Funktion dem Compiler bekanntgemacht werden.

5. Als nächstes folgt das Hauptprogramm:

```
// Hauptprogramm
==============================================================
int main()
{
    init();                         // Ports und Timer 0
initialisieren
    initDisplay();                  // Display aktivieren
    while(1)                        // Start der unendlichen Schleife
    {
        for (OCR0A=MAX_PER; OCR0A>MIN_PER; OCR0A--) // Frequenz
```

```
erhöhen
        {
            _delay_ms(10);                           // in
Schritten von 10ms
        }
        _delay_ms(WAIT_TIME);                        // Wartezeit
hohe Frequenz
        TGL_BIT(PORTB,PB2);
        for (OCR0A=MIN_PER; OCR0A<MAX_PER; OCR0A++)  // Frequenz
absenken
        {
            _delay_ms(10);                           // in
Schritten von 10 ms
        }
        _delay_ms(WAIT_TIME);                        // Wartezeit
niedrige Frequenz
        TGL_BIT(PORTB,PB2);
    }                            // Ende der unendlichen Schleife
}
```

Das Hauptprogramm besteht aus folgenden Teilen:

1. **Initialisierungsteil**: Zu Beginn werden einmalig-abzuarbeitende Programmteile ausgeführt. Darunter fällt insbesondere die Konfiguration der Hardware. Für die Ausgabe eines Signals muss das Direction-Register vorbereitet werden. Zusätzlich muss das Timer-Counter-Modul für die Ausgabe eines Wechselsignals (Pulsweiten-moduliertes Signal, PWM-Signal) vorbereitet werden. Diese wird über die Unterfunktionen `init()` und `initDisplay()` vorgenommen
2. **Endlosschleife**: damit ein Programm vom Microcontroller dauerhaft ausgeführt wird, muss dies in einer Schleife eingebunden sein. Diese wird durch das Konstrukt `while(1){…}` vorgegeben.
3. In der Endlosschleife sind scheinen die Zeilen 77..82 und 84..89 ganz ähnlich auszusehen.
    1. Dort wird zunächst in einer for-Schleife das Register `OCR0A` von der maximalen Periodendauer `MAX_PER` zur minimalen `MIN_PER` heruntergezählt und beim Zählschritt jeweils 10 Millisekunden gewartet (`_delay_ms(10)`). Wie im Video dargestellt, bietet es sich an für die Details zum Output Compare Register (`OCR0A`) den entsprechenden Teil des ATmega88-Datenblatts durchzulesen. Als leichten Einstieg kann auch die deutsche Übersetzung des ATmega88-Datenblatts per Index nach `OCR0A` durchsucht werden.
    2. Nachdem bis zur kürzesten Periode gezählt wurde, soll der höchste Ton die Dauer von `WAIT_TIME` Millisekunden gehalten werden.
    3. Der Zustand der LED soll dann gewechselt werden.
    4. In den Zeilen 84..89 ist das gleiche für eine länger werdende Periodendauer eingefügt. Der einzige Unterschied besteht darin, das in der for-Schleife nun herauf statt herunter gezählt wird.

6. Nach dem Hauptprogramm sind die Unterfunktionen aufgelistet:

```

// Funktionen
=================================================================

```

```
// Generelle Initialisierungsfunktion
void init()
{
    initPorts();                // Ports auf Ausgang schalten
    initTimer();                // Timer zur Sounderzeugung
starten
}

// Initialisierung der I/O-Ports
void initPorts()
{
    DDRB |= (1<<DDB2);            // Port B, Pin 0 (zur LED) auf
Ausgang
    DDRD |= (1<<DDD5);            // Port D, Pin 5 (zum Buzzer)
auf Ausgang
}

// Intialisierung des Timers 0 fuer Sounderzeugung
void initTimer()
{
    TCCR0A = (1<<WGM01) |(1<<COM0B0);     // CTC Mode waehlen und
Toggle Mode nur für OC0B aktivieren
    TCCR0B = (1<<CS01 | 1<<CS00);        // Timer-Vorteiler /64

    OCR0A = MAX_PER;            // Start mit tiefstem Ton
}
```

1. `void init()`: Bei längeren Programmen bietet es sich an eine übergeordnete init-Funktion anzulegen, aus welcher die einzelnen Initialisierungen von Sensoren und ähnlichem aus aufgerufen werden.
2. `void initPorts()`: In dieser Funktion werden die Data Direction Register der Ports B und D korrekt zugewiesen.
3. `void initTimer()`: für das Timer Modul muss der gewünschte “Clear Timer on Compare” Modus und Hardware-Vorteiler gewählt werden.
   Mit dem Teiler $r_{\rm prescaler}=64$ ergibt sich für die Timer-Schritte pro Sekunde:
   \begin{align*} f_{\rm Timer}= f_{\rm Quarz} / r_{\rm prescaler} = 18'432'000 ~\rm Hz / 64 =288'000 ~\rm Hz \end{align*} Da mit `COM0B0 = 0` ein Invertieren des Ausgangs eingestellt wurde, wäre die höchste ausgegebene Frequenz $f_{\rm out, max}= f_{\rm Timer}/2 = 144'000 ~\rm Hz$.
   Diese würde sich ergeben, wenn der Timer angewiesen würde nur einen Schritt zu zählen. Für eine Frequenz von $f_{\rm out}= 1'000~\rm Hz$ muss `OCR0A` wiefolgt gesetzt werden: \begin{align*} {\rm OCR0A}=f_{\rm out, max}/f_{\rm out}-1=144'000 ~\rm Hz / 1'000~\rm Hz - 1= 143 \end{align*} Dies entspricht gerade `MIN_PER = 143`.

7. Ansprechen des Displays:

```


// Initialisierung der Display-Anzeige
void initDisplay()              // Start der Funktion
```

```
{
    lcd_init();                     // Initialisierungsroutine aus der lcd_lib
    lcd_gotoxy(0,0);                    // Cursor auf 1. Zeile, 1. Zeichen
    lcd_putstr("- Experiment 2 -"); // Ausgabe Festtext: 16 Zeichen

    lcd_gotoxy(1,0);                    // Cursor auf 2. Zeile, 1. Zeichen
    lcd_putstr(" Creating Sound ");    // Ausgabe Festtext: 16 Zeichen

}                               // Ende der Funktion
```

In der Funktion `void initDisplay` wird das Display angewiesen Daten auszugeben

1. `lcd_init()`: Diese Unterfunktion sollte immer ausgeführt werden, bevor das Display angesprochen werden soll.
2. `lcd_gotoxy(x,y)`: Diese Funktion weist das Display an die kommende Ausgabe an der Position x,y auszugeben.
3. `lcd_putstr(string)`: Gibt einen vordefinierten Text an der aktuellen Position aus.

IV. Ausführung in Simulide

1. Geben Sie die oben dargestellten Codezeilen nacheinander ein und Kompilieren Sie den Code.
2. Öffnen Sie Ihre hex-Datei in SimulIDE und testen Sie, ob diese die gleiche Ausgabe erzeugt

Sie sollten sich nach der Übung die ersten Kenntnisse mit dem Umgang der Umgebung angeeignet haben. Bitte arbeiten Sie folgende Aufgaben durch:

Aufgaben

1. Klicken Sie mit rechter Maustaste bei `main()` auf `WAIT_TIME` und dann auf Goto implementation. Sie werden feststellen, dass der Cursor auf die Deklaration des Wertes springt. Versuchen Sie selbiges bei `initDisplay` an `lcd_putstr` und wählen Sie die Variante, welche ein **{…}** angefügt hat. Hier sehen Sie die den Code in der header-Datei, der für die Übergabe des Strings an das Display verantwortlich ist. Dort kann in gleicher Art `lcd_putc` und `lcd_write` weiterverfolgt werden. In `lcd_write` und `lcd_enable` wird die Übergabe der Werte an das Display abgearbeitet. Dazu werden zunächst die Daten am Datenport ausgegeben und anschließend die Steuerleitung gepulst aktiviert. Verfolgen Sie `lcd_gotoxy` nach. Wie wird das übertragen?
2. Wieso wird an Pin B5 überhaupt was ausgegeben? Im Programm wird doch nirgends PORTB gesetzt, oder?
    1. Versuchen Sie mit dem Datenblatt (Seite 75ff insbesondere Kapitel 12.4) herauszufinden wie das funktioniert.
    2. Prüfen Sie dazu im Datenblatt auch nach, was an Pin B5 noch für Funktionen

hängen (Kapitel 1. Pin Configurations).
    3. Warum wird zwar `OCR0`**A** als Vorgabe genutzt, aber `OC0`**B** als Ausgabe gewählt (siehe Zeile: `TCCR0A = (1<<WGM01) |(1<<`**COM0B0**`))`?
3. Wie ist es möglich bei aufsteigender und abfallender Frequenz einen entsprechenden Text am Display auszugeben? Ändern Sie den Code geeignet.
4. Versuchen Sie das Programm so zu variieren, dass es ein Martinshorn ausgibt. Suchen Sie dazu zunächst die benötigten Frequenzen und ändern Sie das Programm passend ab.
5. Können Sie eine kleine Melodie ausgeben? Versuchen Sie z.B. “Alle meine Entchen”, oder eine Melodie ihrer Wahl.

- Diese Falstad Schaltung skizziert die Struktur des Timer/Counters

From:
https://wiki.mexle.org/ - **MEXLE Wiki**

Permanent link:
https://wiki.mexle.org/microcontrollertechnik/2_sound_und_timer?rev=1699533048

Last update: **2023/11/09 13:30**