

3 Logische Funktionen

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

3. Up-Down Counter

2

Tasten einlesen

2

Ziele

2

Video

2

Übung

2

To Do: notwendige header-Dateien

10

3. Up-Down Counter

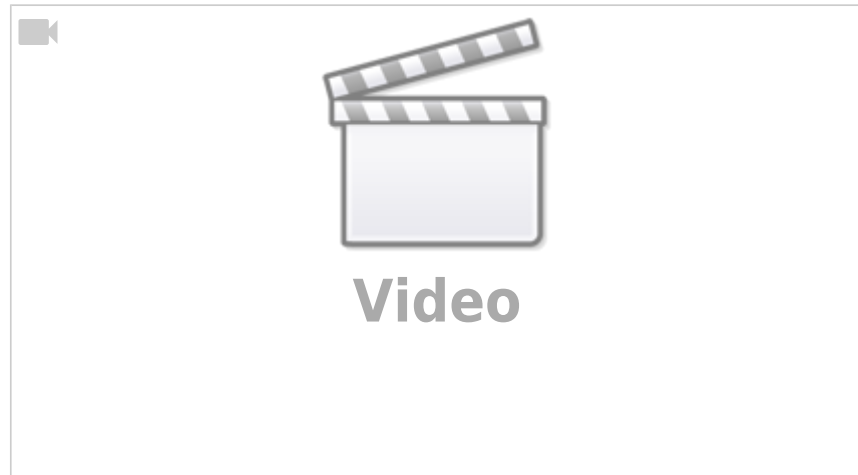
Tasten einlesen

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie eine Taste eingelesen werden kann

Video



Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
 1. [3_logic_functions.simu](#)
 2. [3_logic_functions.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 `3_logic_functions.simu`
 2. Laden Sie `3_logic_functions.hex` als firmware auf den 328 Chip
2. Betrachtung der neuen Komponenten: In der Simulation sind nun neben dem Microcontroller, der LED und dem Display Hd44780 zwei Schalter als neue Komponenten zu sehen, welche mit S1 und S2 bezeichnet sind. Diese werden in diesen Beispiel zur Eingabe genutzt.
3. Betrachtung des Programmablaufs
 1. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
 2. Als nächstes ist ein Displaybild zu sehen, in dem verschiedene logische Formeln mit Ergebnissen abgebildet sind:
 1. AND-Verknüpfung: $S1 \& S2$,
 2. OR-Verknüpfung: $S1 + S2$,
 3. NOT-Verknüpfung: $\neg S1$,
 4. XOR-Verknüpfung: $S1 \text{ xor } S2$
 3. Werden die Tasten S1 und S2 gedrückt, so werden die Ergebnisse aktualisiert.
4. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Eingabe in Atmel Studio

```

/*=====
=====
/*===== =
=====
=====
Experiment 3:  Logische
Basisfunktionen in Software
=====
=====
=====

Dateiname:
Logic_Functions.c

Autoren:      Peter
Blinzinger
              Prof. G.
Gruhler (Hochschule
Heilbronn)
              D.
Chilachava    (Georgische
Technische Universitaet)

Version:      1.2 vom
27.04.2020

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 dem
Display werden Ergebnisse
von
              logischen
Verknuepfungen (UND, ODER,
NOT, XOR) dargestellt.
              Die
logischen Eingangssignale
werden von den Tasten S1 und
S2
              eingelesen.

```

Ändern Sie auch hier wieder die Beschreibung am Anfang des C-Files, je nachdem was Sie entwickeln

Deklarationen

```

=====

```

1. Hier wird wieder nach dem Quarz geprüft und ggf. dessen Frequenz eingestellt
2. Bei den Header-Dateien wird nun die `stdbool.h` Datei inkludiert. Über diese können die Funktionen der booleschen Algebra genutzt werden.
3. Als Konstanten werden NULL und EINS definiert. Dieser hexadezimalen Zahlencode 0x30 und 0x31 entsprechen ausgebenbare Zeichen nach dem [ASCII](#) Standard. Der ASCII Standard gibt für jedes darstellbare Zeichen einen Code vor. In [figure 1](#) ist die ASCII Tabelle gezeigt. Dort ist **horizontal** die erste Zahl (z.B. 0x**30**) und **vertikal** die zweite Zahl

```

Displayanzeige: Start (fuer
2s):          Betrieb:
                +-----+
-----+      +-----+
---+
                | -
Experiment 3 - |
|S1&S2=0  S1+S2=0|
                |Logic
Functions |      | /S1=1
S1xorS2=0|
                +-----+
-----+      +-----+
---+

```

Tastenfunktion: S1 und S2
sind die Logikeingänge.
Betrieb ohne Entprellung

Jumperstellung: keine
Auswirkung

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

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

```

=====
=====
=====*/

```

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

```

// Festlegung der
Quarzfrequenz
#ifndef F_CPU
// optional definieren
#define F_CPU 12288000UL
// MiniMEXLE mit 12,288 MHz
Quarz
#endif

```

```

// Include von Header-
Dateien
#include <avr/io.h>

```

(0x30) aufgetragen. Diese führen zu den
darstellbaren Zahlen '0' und '1'.

- Die Variablen sw1 und sw2 sollen im
Folgenden den Zustand des Schalters
anzeigen.
- Die Makros wurden bereits erklärt
- Die Funktionsprototypen zeigen wieder die
kommenden Unterprogramme an

Fig. 1: ASCII Tabelle

ASCII Code Chart															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOI	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
6	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n
7	p	q	r	s	t	u	v	w	x	y	z	{		}	DEL

Hauptprogramm =====

- Zunächst werden zwei Initialisierungsroutinen
aufgerufen (siehe weiter unten)
- Dann wird eine temporäre Variable deklariert,
welche im Folgenden die das ASCII-Zeichen
der Ergebnisse enthält
- In der Endlosschleife wird zunächst die
Unterfunktion readButtons() aufgerufen
(siehe weiter unten)
- die Zeilen 84...102 scheinen sich sehr zu
ähneln:
 - Hier steht jeweils zuerst eine if-
Anweisung. In Abhängigkeit von der
jeweiligen booleschen Funktion wird die
temporäre Variable gleich NULL (also
das Zeichen '0') oder EINS ('1') gesetzt.
 - Die Funktion lcd_gotoxy(0,6)
versetzt wieder die Position am Display
und lcd_putc(temp) gibt die
temporäre Variable aus
- Für die verschiedenen booleschen Funktionen
steht jeweils eine if-Anweisung bereit. Auch
die Position am Display ist abhängig von der
booleschen Funktion.
- In Zeile 104 wird dann eine gewisse Zeit
gewartet. Dies vermeidet das "Prellen" des
realen Schalters: In Realität wird bei

```
// I/O Konfiguration (intern
// weitere Dateien)
#include <util/delay.h>
// Definition von Delays
// (Wartezeiten)
#include <stdbool.h>
// Bibliothek fuer Bit-
// Variable
#include "lcd_lib_de.h"
// Funktionsbibliothek zum
// LCD-Display

// Konstanten
#define NULL    0x30
// ASCII-Zeichen '0'
#define EINS    0x31
// ASCII-Zeichen '1'

// Variable
bool sw1 = 0;
// Bitspeicher fuer Taste 1
bool sw2 = 0;
// Bitspeicher fuer Taste 2

// Makros
#define SET_BIT(PORT, BIT)
((PORT) |= (1<<BIT)) //
Einzelbit auf Port SET
#define CLR_BIT(PORT, BIT)
((PORT) &= ~(1<<BIT)) //
Einzelbit auf Port RESET

// Funktionsprototypen
void initDisplay(void);
// Initialisierung Display
// und Startanzeige
void initTaster(void);
// Initialisierung der
// Taster
void readButtons(void);
// Einlesen der Tastenwerte

// Hauptprogramm
=====
=====
=====
int main()
// Start des Hauptprogramms
{
    initDisplay();
// Initialisierung Display
```

Tastendruck nicht nur einmal der Kontakt geschlossen, sondern häufig mehrmals. Dies kann aber zu fehlerhaften Zuständen führen.

Funktionen =====

1. In `initDisplay` wird wieder zunächst das Display initialisiert und die Startanzeige mit dem Namen des Programms angezeigt
2. Nach 2 Sekunden werden dann die booleschen Funktionen auf dem Display dargestellt. Dort sind die Ergebnisse für nicht gedrückte Schalter vorgegeben.
3. Funktion `initTaster`
 1. Durch die Änderung des Datenrichtungs-Register (DDR) wird die Richtung der Anschlüsse vorgegeben. Diese beiden Anschlüsse sind in der Simulide Umgebung an die Schalter S1 und S2 angeschlossen. Durch die UND-Verknüpfung mit der Maske `0b11111001` werden die Anschlüsse B0 und B3..B7 nicht geändert, sondern nur die Anschlüsse B1 und B2 auf Eingang gesetzt.
 2. Mit der Zuweisung von `PORTB` wurde bisher bei Ausgängen der Ausgabewert vorgegeben. Bei Eingängen wird über diese Zuweisung ein [Pullup-Widerstand](#) dazugeschalten. Damit ergibt sich aus dem äußeren Schalter und dem internen Widerstand ein Spannungsteiler. Bei leitfähigem Schalter gibt der Spannungsteiler \$0V\$ (=logisch \$0\$) zum Microcontroller aus, bei offenem Schalter \$5V\$ (=logisch \$1\$).
4. Funktion `readButtons` liest aus dem Register `PINB` das dem Schalter entsprechende Bit aus. Als Eselsbrücke: `PIN` steht für Input, `PORT` für Output.

```
    initTaster();  
    // Initialisierung der  
    Buttons  
    unsigned char temp;  
    // temporaere Variable  
    definieren  
  
    while(1)  
    // unendliche Schleife  
    {  
  
        readButtons();  
    // aktuelle Tastenwerte  
    einlesen  
        if (sw1&&sw2)  
temp=EINS; // Ergebnis der  
UND-Verknuepfung  
        else temp=NULL;  
        lcd_gotoxy(0,6);  
        lcd_putc(temp);  
    // auf LCD als Zeichen 0  
    oder 1 ausgeben  
  
        if (sw1||sw2)  
temp=EINS; // Ergebnis der  
ODER-Verknuepfung  
        else temp=NULL;  
        lcd_gotoxy(0,15);  
        lcd_putc(temp);  
    // auf LCD als Zeichen 0  
    oder 1 ausgeben  
  
        if (!sw1) temp=EINS;  
    // Ergebnis der Negation  
        else temp=NULL;  
        lcd_gotoxy(1,4);  
        lcd_putc(temp);  
    // auf LCD als Zeichen 0  
    oder 1 ausgeben  
  
        if (sw1^sw2)  
temp=EINS; // Ergebnis  
der XOR-Verknuepfung  
        else temp=NULL;  
        lcd_gotoxy(1,15);  
        lcd_putc(temp);  
    // auf LCD als Zeichen 0  
    oder 1 ausgeben  
  
        _delay_ms(100);  
    // Wartezeit 100 ms
```

```
    }  
    // Ende der unendlichen  
    Schleife  
  
}  
// Ende des Hauptprogramms  
  
// Funktionen  
=====
```

```
=====
```

```
=====
```

```
// Initialisierung Display-  
Anzeige  
void initDisplay(void)  
{  
    lcd_init();  
    // Initialisierungsroutine  
    aus der lcd_lib  
    lcd_gotoxy(0,0);  
    // Cursor auf 1. Zeile, 1.  
    Zeichen  
    lcd_putstr("- Experiment  
3 -"); // Ausgabe Festtext:  
    16 Zeichen  
  
    lcd_gotoxy(1,0);  
    // Cursor auf 2. Zeile, 1.  
    Zeichen  
    lcd_putstr("Logic  
Functions "); // Ausgabe  
    Festtext: 16 Zeichen  
  
    _delay_ms(2000);  
    // Wartezeit 2 s  
  
    lcd_gotoxy(0,0);  
    // Cursor auf 1. Zeile, 1.  
    Zeichen  
    lcd_putstr("S1&S2=0  
S1+S2=0"); // Ausgabe  
    Festtext: 16 Zeichen  
  
    lcd_gotoxy(1,0);  
    // Cursor auf 2. Zeile, 1.  
    Zeichen  
    lcd_putstr("/S1=1  
S1xorS2=0"); // Ausgabe  
    Festtext: 16 Zeichen  
}
```

```
// Initialisierung der
Taster
void initTaster(void)
// Bitposition im
Register:
{
    //      ___76543210
    DDRB = DDRB &
0b11111001; // Port B
auf Eingabe schalten
    PORTB =
0b00000110; //
Pullup-Rs eingeschaltet

    _delay_us(10);
// Umschalten der Hardware-
Signale abwarten
}

// Tastenwerte S1 und S2
(ohne Entprellen) einlesen
void readButtons(void)
{
    sw1 = !(PINB & (1 <<
PB1)); // Tasten
invertiert in Bitspeicher
einlesen
    sw2 = !(PINB & (1 <<
PB2)); // somit gedruckte
Taste ="1"
}
```

1. Funktion: Die Funktion des Programms sollte kurz erklärt werden. Damit wird dem Leser bereits vor dem Code schon Hinweise gegeben
2. Display-Anzeige: Ähnlich der Funktion ist auch eine Darstellung der (erwartbaren) Anzeige sinnvoll.
3. Tastenfunktion: Bei zukünftigen Anwendungen kann eine Eingabe von Tastenstellungen sinnvoll sein. Dies sollte hier angegeben werden
4. 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.
5. Fuses im uC: Beim Microcontroller (auch μ C 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
6. Header-Files: Werden weitere Softwareteile genutzt, so sollten diese über [Header-Dateien](#) eingebunden. Diese sollten bereits schon vor dem eigentlichen Codeteil kurz erklärt werden.
7. Nach der Beschreibung steht im Code der Deklarationsbereich:

```
// Deklarationen
```



```

=====

// Festlegung der Quarzfrequenz
#ifndef F_CPU // optional definieren
#define F_CPU 1228800UL // MiniMEXLE mit 12,288 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 59 // minimale Periodendauer in
// "Timerticks"
#define MAX_PER 255 // maximale Periodendauer in
// "Timerticks"
#define WAIT_TIME 2000 // Wartezeit zwischen Flanken in
// ms

// Makros
#define SET_BIT(PORT, BIT) ((PORT) |= (1 << (BIT))) // Port-
// Bit Setzen
#define CLR_BIT(PORT, BIT) ((PORT) &= ~(1 << (BIT))) // Port-Bit
// Loeschen
#define TGL_BIT(PORT, BIT) ((PORT) ^= (1 << (BIT))) // Port-Bit
// Toggeln

// 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

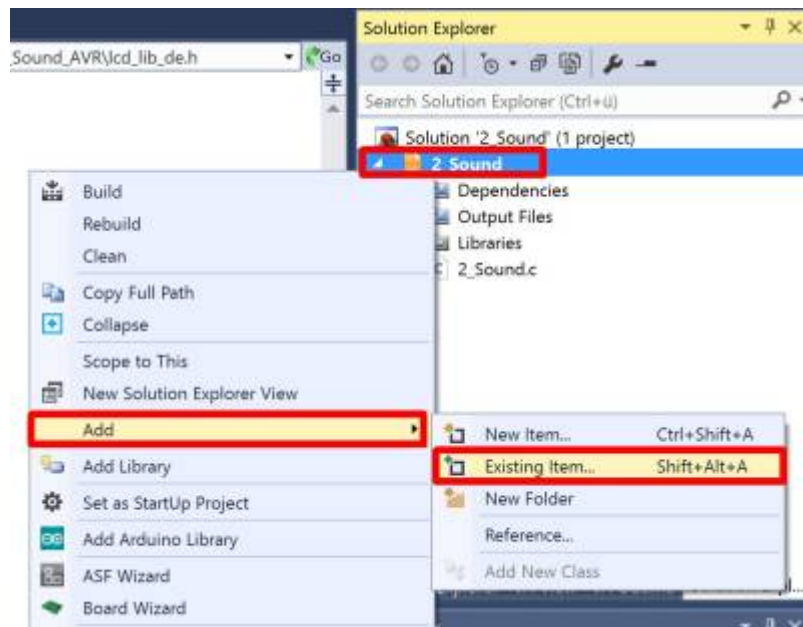
```

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:

8. 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.

9. 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 Beispiel [1._hello_blinking_world](#) verwendet.
 2. `<util/delay.h>`: Header-Datei, welche einen einfachen Umgang mit Wartezeiten ermöglicht. Diese wurde bereits schon im Beispiel [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`) » 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.

10. `#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 Port zu setzen (`SET_BIT(PORT, BIT)`),

2. ein Bit in einem Port zu löschen (CLR_BIT(PORT, BIT)), oder
3. ein Bit in einem Port zu invertieren (TGL_BIT(PORT, BIT)).
11. Funktionsprototypen: Wie regulär bei der C-Programmierung sollten die verwendeten Funktion dem Compiler bekanntgemacht werden.
12. 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,DDB0);

        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,DDB0);
    }                             // Ende der unendlichen Schleife
}
```

Das Hauptprogramm besteht aus folgenden Teilen:

13. **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
14. **Endlosschleife:** damit ein Programm vom Microcontroller dauerhaft ausgeführt wird, muss dies in einer Schleife eingebunden sein. Diese wird durch das Konstrukt `while(1){...}` vorgegeben.
15. 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 [ATmega328-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.
16. 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<<DDB0);     // 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
    TCCR0B = (1<<CS01 | 1<<CS00);     // Timer-Vorteiler /64

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

17. `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.
18. `void initPorts()`: In dieser Funktion werden die Data Direction Register der Ports B und D korrekt zugewiesen.
19. `void initTimer()`: für das Timer Modul muss der gewünschte "Clear Timer on Compare" Modus und Hardware-Vorteiler gewählt werden. Mit dem Teiler $f_{\text{prescaler}}=64$ ergibt sich für die Timer-Schritte pro Sekunde: $f_{\text{Timer}} = f_{\text{Quarz}} / r_{\text{prescaler}} = 12'288'000 \text{ Hz} / 64 = 192'000 \text{ Hz}$. Da mit `COM0B0=0` ein

Invertieren des Ausgangs eingestellt wurde, wäre die höchste ausgegebene Frequenz $f_{\text{out, max}} = f_{\text{Timer}}/2 = 96'000 \text{ Hz}$. Diese würde sich ergeben, wenn der Timer angewiesen würde nur einen Schritt zu zählen. Für eine Frequenz von $f_{\text{out}} = 1'600 \text{ Hz}$ muss $\text{OCR0A} = f_{\text{out, max}}/f_{\text{out}} - 1 = 96'000 \text{ Hz} / 1'600 \text{ Hz} - 1 = 59$ gesetzt werden. Dies entspricht gerade $\text{MIN_PER} = 59$.

20. 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

21. `lcd_init()`: Diese Unterfunktion sollte immer ausgeführt werden, bevor das Display angesprochen werden soll.
22. `lcd_gotoxy(x,y)`: Diese Funktion weist das Display an die kommende Ausgabe an der Position x,y auszugeben.
23. `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. Wie ist es möglich bei aufsteigender und abfallender Frequenz einen entsprechenden Text am Display auszugeben? Ändern Sie den Code geeignet.
3. 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.
4. Können Sie eine kleine Melodie ausgeben? Versuchen Sie z.B. "Alle meine Entchen", oder eine Melodie ihrer Wahl.

From:

<https://wiki.mexle.org/> - **MEXLE Wiki**

Permanent link:

https://wiki.mexle.org/microcontrollertechnik/3_logische_funktionen?rev=1588807034

Last update: **2021/05/09 10:08**

