

Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion (Software)

Studienarbeit I

des Studienganges Elektrotechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

von
Johannes Schackniß

05.04.2021

Bearbeitungszeitraum:	18.01.2021 - 05.04.2021
Matrikelnummer, Kurs:	4175962, TEL18AAT
Ausbildungsfirma:	ABB Automation GmbH
Betreuer der Dualen Hochschule:	Prof. Dr.-Ing. Michael Ullmann

Erklärung

Ich, Johannes Schackniß, versichere hiermit, dass ich meine Projektarbeit mit dem Thema: „Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion (Software)“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift (Johannes Schackniß)

Abstract

Diese Studienarbeit umfasst die Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion (Software). Bei einem Lichtwecker erfolgt das Wecken des Anwenders mit Hilfe von Licht. Hierbei wird der Einfluss von Licht, auf den Biorhythmus des Menschen, genutzt. Im ersten Teil der Arbeit, der theoretischen Vorbetrachtung, sind die einzelnen Softwarebibliotheken und Hardwarekomponenten thematisiert. Die Umsetzung ist mit einem ESP32 realisiert, wobei die Zeit über das Network Time Protocol synchronisiert wird. Zunächst erfolgt die Implementierung der Einzelkomponenten. Anschließend werden eine Datenstruktur, sowie eine akustische Weckfunktion und eine Lichtweckfunktion, implementiert. Die Einzelkomponenten und Weckfunktionen sind über eine Menüstruktur zusammengeführt und innerhalb der `loop()`-Funktion logisch miteinander verknüpft. Der Lichtwecker verfügt über vier einstellbare Wecker. Das akustische Wecksignal ist mit einem MP3 Player, und die Lichtweckfunktion mit einer Neopixel Matrix, umgesetzt. Sowohl der MP3 Player, als auch die Neopixel Matrix sind eigenständig einsetzbar.

This project thesis covers the software development of a light alarm clock with clock and alarm functionality. A light alarm clock uses light to wake up the user. For this purpose the influence of light on the human biorhythm is used. In the first part of the thesis the theoretical preliminary considerations, the individual software libraries and hardware components are discussed. The implementation is realized with an ESP32, the synchronization of the time is done via the network time protocol. First of all the implementation of the individual components takes place. Then a data structure, as well as an acoustic alarm function and a light alarm function are implemented. The individual components and alarm functions are combined in a menu structure and logically linked within the `loop()`-function. The light alarm clock has four adjustable alarms. The acoustic alarm is implemented with an MP3 player and the light alarm with a neopixel matrix. Both the MP3 player and the neopixel matrix can be used independently.

Inhaltsverzeichnis

Erklärung	II
Abstract	III
Inhaltsverzeichnis	IV
Abbildungsverzeichnis	VII
Tabellenverzeichnis	IX
Listings	X
Abkürzungsverzeichnis	XI
1. Einleitung	1
2. Aufgabenstellung	2
3. Geplante Vorgehensweise	3
4. Theoretische Vorbetrachtung	5
4.1. Biorhythmus	5
4.2. Mikrocontroller	5
4.2.1. Arduino Mikrocontroller	6
4.2.2. Espressif Systems Mikrocontroller	7
4.3. Zeit und Zeitsynchronisation	7
4.3.1. Zeitstandards	8
4.3.2. Zeitsynchronisationsmechanismen	8
4.4. Display	9
4.5. Beleuchtung	10
4.6. Bedienelemente	12
4.7. Tonausgabe	14

4.8. Echtzeit	16
4.9. Vergleichbare Projekte	17
5. Lösungsbeschreibung und Umsetzung	19
5.1. Verwendete Hardware	19
5.2. Aufbau der Software	20
5.3. Implementierung der einzelnen Komponenten	21
5.3.1. Zeit und Zeitsynchronisation	22
5.3.2. Wireless Local Area Network (WLAN) Manager	23
5.3.3. Bedienelemente - Taster	24
5.3.4. Liquid Crystal Display (LCD)	26
5.3.5. Beleuchtung - lichtemittierende Diode (LED) Panel	27
5.3.6. Tonausgabe - Moving Picture Experts Group Audio Layer 3 (MP3) Player	28
5.4. Weckfunktionen	29
5.4.1. Datenstruktur	29
5.4.2. Akustische Weckfunktion	30
5.4.3. Lichtweckfunktion	31
5.4.4. Beenden der Wecksignale	33
5.5. Zusammenführung der Einzelkomponenten und Weckfunktionen	34
5.5.1. Menüstruktur	34
5.5.2. loop()-Funktion	35
5.5.3. Test der zyklischen Ausführungszeit	36
6. Zusammenfassung	40
6.1. Fazit	40
6.2. Ausblick	41
Literaturverzeichnis	43
A. Anhang: Quellcode des Lichtweckers	49
A.1. 2021-03-27_V28.ino	49
A.2. alarm.ino	56
A.3. backEvent.ino	58
A.4. light.ino	61
A.5. ntp.ino	62
A.6. other.ino	64
A.7. selectEvent.ino	65
A.8. setup.ino	77

A.9. writeDisplay.ino	81
B. Anhang: Menüstruktur	86
C. Anhang: loop()-Ablaufdiagramm	87
D. Anhang: digitaler Anhang	88

Abbildungsverzeichnis

3.1. Projektplan - Gantt-Diagramm	3
4.1. Arduino DUE [10]	6
4.2. Arduino MEGA [11]	6
4.3. ESP32 in der Ausführung ESP32 NodeMCU von AZ-Delivery [14]	7
4.4. 0,96 Zoll Organic Light Emitting Diode (OLED) Inter-Integrated Circuit (I2C) Display 128x64 Pixel [20]	9
4.5. HD44780 2004 LCD Bundle 4x20 Zeichen mit I2C Schnittstelle [23]	10
4.6. LED Rot-Grün-Blau (RGB) Modul [25]	11
4.7. Neopixel Ring [29]	12
4.8. Neopixel Matrix [30]	12
4.9. Taster [31]	12
4.10. Drehwinkelgeber [34]	13
4.11. Abtastung Drehwinkelgeber [32, S. 3]	13
4.12. Prellen eines mechanischen Kontaktes [37]	13
4.13. Taster mit Pull-Up Widerstand [8]	14
4.14. Taster mit Pull-Down Widerstand [8]	14
4.15. Buzzer Modul aktiv [40]	15
4.16. Buzzer Modul passiv [41]	15
4.17. MP3 Player Modul [45]	16
4.18. Wertfunktionen zur Bewertung von Zeitbedingungen [46, S. 322]	17
5.1. Ablaufdiagramm zum Aufbau der Software	21
5.2. Access Point „Lichtwecker Setup“	23
5.3. Webinterface: Lichtwecker Setup	24
5.4. Webinterface: Configure WiFi	24
5.5. Ablaufdiagramm einer Interrupt Service Routine (ISR)	26
5.6. Ablaufdiagramm zur Ansteuerung des LED Panels	28
5.7. Ablaufdiagramm der Funktion <code>wakeUpTime(wecker wecker)</code>	31

5.8. Ablaufdiagramm der Funktion `wakeUp()` 31

5.9. Ablaufdiagramm der Funktion `wakeUpLight(wecker wecker)` 32

5.10. Ablaufdiagramm der Funktion `alarmOff()` 33

5.11. Histogramm zur Ausführungszeit der `loop()`-Funktion 39

B.1. Menüstruktur 86

C.1. Ablaufdiagramm der `loop()`-Funktion 87

Tabellenverzeichnis

5.1. Externe Interrupts und dazugehörige ISRs	25
5.2. Datenstruktur <code>struct wecker</code>	29
5.3. Laufzeit der <code>loop()</code> -Funktion bei 1000000 Durchläufen	38

Listings

5.1. Statische WLAN-Verbindung	22
5.2. Ausgabe von vier Zeilen Text auf LCD	26
5.3. Messung und Ausgabe der Ausführungszeit von <code>loop()</code>	37
A.1. 2021-03-27_V28.ino	49
A.2. alarm.ino	56
A.3. backEvent.ino	58
A.4. light.ino	61
A.5. ntp.ino	62
A.6. other.ino	64
A.7. selectEvent.ino	65
A.8. setup.ino	77
A.9. writeDisplay.ino	81

Abkürzungsverzeichnis

BLE	Bluetooth Low Energy
bzgl.	bezüglich
bzw.	beziehungsweise
DNS	Domain Name System
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPIO	General Purpose Input/Output Interface
I2C	Inter-Integrated Circuit
I2S	Inter-Integrated Circuit Sound
IDE	integrierte Entwicklungsumgebung
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISR	Interrupt Service Routine
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	lichtemittierende Diode
microSD	Micro Secure Digital Memory Card
MEZ	mitteleuropäische Zeit
MESZ	mitteleuropäische Sommerzeit
MP3	Moving Picture Experts Group Audio Layer 3
NTP	Network Time Protocol
OLED	Organic Light Emitting Diode
PWM	Pulsweitenmodulation
RGB	Rot-Grün-Blau
RTC	Real-Time Clock
SPI	Serial Peripheral Interface
TAI	Temps Atomique International
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol

UTC Coordinated Universal Time
WLAN Wireless Local Area Network
z.B. zum Beispiel

1. Einleitung

„Der Mensch verbringt rund ein Drittel seines Lebens im Schlaf. Zweifelsfrei gilt, dass Schlaf die durch den Wachzustand hervorgerufene Ermüdung beseitigt und eine lebensnotwendige Aufbau- und Erholungsphase darstellt.“[1] Während des Schlafens durchläuft jeder Mensch mehrere Phasen. Die letzte Phase, kurz vor dem Aufwachen, ist das Postdormitium.

„Das Postdormitium, der Übergang zwischen Schlaf und Wachen, tritt [vor allem] bei verlängerter Schlafzeit auf“[1], jedoch ist diese Schlafphase, sowie der Schlaf-wach-Rhythmus, durch die Verwendung eines Weckers gestört. Hierbei wird die Schlafphase durch einen akustischen Weckton beendet, sodass der Mensch in den Wach-Zustand übergeht. Dieser abrupte Wechsel zwischen Schlaf und Wach kann zu einer Minderung der Stimmung und des Allgemeinbefindens führen. Somit ist diese Weckmethode nicht optimal.

„Der Schlaf-wach-Rhythmus ist eingebettet in das Zusammenspiel verschiedener biologischer Rhythmen.“[1] „[Der menschliche zirkadianer] Schrittmacher [...] synchronisiert [die] inneren biologischen Rhythmen mit dem Tagesverlauf des Sonnenlichts [...].“[2] Dies ist nutzbar, um eine angenehmere Weckmethode zu gestalten, da „Licht auch als effektive, nichtinvasive Therapieoption mit geringen bis keinen Nebenwirkungen eingesetzt werden [kann], um Schlaf, Stimmung und Allgemeinbefinden zu verbessern“[2].

Den Einfluss auf den biologischen Rhythmus nutzen Lichtwecker. Dabei handelt es sich um Wecker, die am Morgen die Lichtintensität erhöhen und somit den menschlichen Körper stimulieren. Der künstliche Sonnenaufgang führt zu einem sanften Übergang vom Schlaf in den Wachzustand. Um das Aufwachen zu einer bestimmten Uhrzeit zu gewährleisten, verfügen Lichtwecker, wie herkömmliche Wecker, über eine akustische Weckfunktion. [3]

Die Entwicklung eines soeben beschriebenen Lichtweckers ist Thema dieser Arbeit, wobei der Fokus auf der Softwareentwicklung liegt.

2. Aufgabenstellung

Die Studienarbeit thematisiert die Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion und ist im Zeitraum von 11 Wochen umzusetzen. Diese Ausarbeitung legt den Fokus auf die Softwareentwicklung und ist Teil eines Projekts, welches aus zwei Arbeiten besteht. Hier ist anzumerken, dass die Umsetzung in enger Zusammenarbeit mit dem Studierenden Robin Gilg erfolgt, der die Hardwareentwicklung [4] bearbeitet. Bei der Softwareentwicklung des Lichtweckers sind folgende Funktionen im Verlauf der Studienarbeit umzusetzen:

- Anzeige der aktuellen Uhrzeit und des Datums in einem Display
- mehrere programmierbare Alarmzeiten
- akustischer Alarm
- Lichtweckfunktion
- Lichtquelle als Nachtlicht
- automatisierte Aktualisierung von Datum und Uhrzeit

Zu den oben genannten Funktionen des Lichtweckers kommt hinzu, dass die programmierbaren Alarmzeiten über eine Wochenendabschaltung verfügen sollen. Des Weiteren ist die Lichtweckfunktion an den menschlichen Biorhythmus anzupassen, wobei Farbe und Helligkeit wechseln können und die Parameter einstellbar sind. Die für die Lichtweckfunktion verwendete Lichtquelle soll zudem als normales Nachtlicht mit einstellbarer Farbe und Helligkeit nutzbar sein.

Optional ist der akustische Alarm mit einem MP3-Player Modul umzusetzen. Somit ist ein beliebiger Weckton wählbar und der Lichtwecker ist auch als eigenständiger MP3-Player nutzbar. Wenn bei der automatisierten Aktualisierung von Datum und Uhrzeit eine Verbindung zu einem WLAN hergestellt wird, ist eine Möglichkeit zur Einstellung der Zugangsdaten über ein WLAN-fähiges Endgerät zu integrieren.

3. Geplante Vorgehensweise

Zu Beginn des Projektes erfolgt zunächst die Projektplanung. Diese ist mit Hilfe eines in Microsoft Visio erstellten Gantt-Diagramms realisiert, welches in Abbildung 3.1 zu sehen ist. Das Wasserfallmodell kommt als Prozessmodell zum Einsatz, da die in Kapitel 2 beschriebenen Anforderungen und Zielsetzungen bereits zu Beginn des Projektes vollständig bekannt sind. Die Tätigkeiten von Abbildung 3.1 sind aufeinander aufbauend, sodass die Bearbeitung sequenziell erfolgt. Hierbei sind die zu Beginn definierten Anforderungen das antreibende Moment, mit dem Ziel den Projektmanagementaufwand möglichst gering zu halten. [5, S. 201 f.]

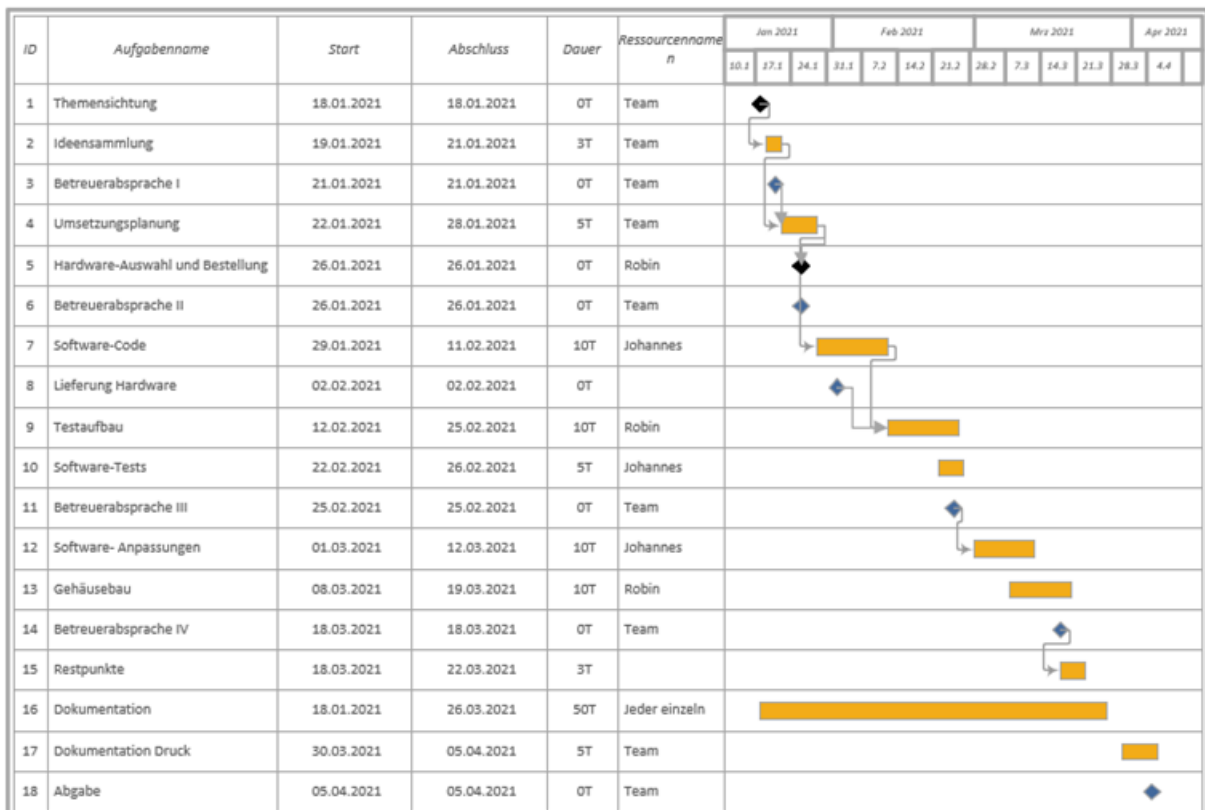


Abbildung 3.1.: Projektplan - Gantt-Diagramm

Nach Abschluss der Projektplanung erfolgt die Einarbeitung in die Einzelkomponenten, um diese zu evaluieren und anschließend auszuwählen. Die Evaluierung des Mikrocontrollers, der Lichtquelle, der Bedienelemente, des Displays und der Umsetzung des akustischen Alarms erfolgt in enger Zusammenarbeit mit dem Studenten Robin Gilg, der die Hardwareentwicklung des Lichtweckers in [4] bearbeitet. Weiterer Teil der theoretischen Vorbetrachtung ist der menschliche Biorhythmus und Schlaf-Wach-Rhythmus, um die Funktionsweise eines Lichtweckers besser zu verstehen. Da die Anzeige von Uhrzeit und Datum, sowie deren automatische Aktualisierung erforderlich ist, sind in der Vorbetrachtung verschiedene Umsetzungsmöglichkeiten zu diskutieren. Daraufhin folgt die Betrachtung, der für den Lichtwecker erforderlichen, Echtzeitkriterien.

Sobald die theoretische Vorbetrachtung beendet ist, folgt die Umsetzung der in Kapitel 2 definierten Anforderungen. Hierfür sind zunächst die einzelnen Hardwarekomponenten zu implementieren. Somit ist gewährleistet, dass die einzelnen Bauteile und Funktionen eigenständig funktionieren, bevor die Zusammenführung zu einem Lichtwecker stattfindet. Nach der Implementierung der Hardwarekomponenten folgt die Umsetzung der Weckfunktion. Dies umfasst die Entwicklung einer Datenstruktur, sowie die Umsetzung der Licht- und Musikweckfunktion.

Final sind die Einzelkomponenten und die Lichtweckfunktion zusammenzuführen. Dies beinhaltet unter anderem die Entwicklung einer Menüstruktur zur Ausgabe auf einem Display, sowie die logische Verknüpfung der Einzelkomponenten und Weckfunktionen über die `loop()`-Funktion. Mit diesem finalen Schritt ist die Software des Lichtweckers erstmals voll funktionsfähig. Somit folgt anschließend das Testen zusammen mit der Hardware, als auch der zyklischen Ausführungszeit, um die Einhaltung der Echtzeitkriterien zu prüfen.

4. Theoretische Vorbetrachtung

Dieses Kapitel der Studienarbeit beinhaltet die theoretische Vorbetrachtung zum Thema „Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion (Software)“. Dies umfasst eine allgemeine Betrachtung des menschlichen Biorhythmus, als auch die Auseinandersetzung mit den verschiedenen Hardwarekomponenten und deren Ansteuerung mit Hilfe von Software. Zudem sind die Themen Zeit und Zeitsynchronisation, sowie Echtzeit und vergleichbare Projekte beschrieben.

4.1. Biorhythmus

„Chronobiologie ist die Lehre von der zeitlichen Organisation biologischer Systeme und Prozesse in Bezug auf Physiologie und Verhalten. Oder vereinfacht ausgedrückt, die Wissenschaft von der inneren Uhr.“[6, S. 32] „Die biologische Uhr bestimmt, [...], zu welchem Zeitpunkt wir müde werden und aufwachen [...]. Licht ist der wichtigste Zeitgeber für die Synchronisation der biologischen Uhr mit der Umwelt.“[6, S. 175] Im speziellen ist „[die] Sonne [...] vor allem für die Synchronisation der Phasenlänge verantwortlich.“[6, S. 34]

4.2. Mikrocontroller

„Der Mikrocontroller ist ein hochintegrierter Baustein, der, neben einem Standard - Mikroprozessor, einen oder mehrere serielle und parallele Portbausteine, Datenspeicher, Programmspeicher und eventuell noch einige andere Sonderfunktionen an Hardware auf einem einzigen Chip vereinigt.“[7, S. 2]

Hauptsächlich unterscheiden sich Mikrocontroller einerseits durch die Breite des Befehlsregisters, den Befehlssatz und die Leistungsfähigkeit des Prozessors. [8] Andererseits findet auch eine Unterscheidung nach Schnittstellen und der Anzahl von General Purpose Input/Output Interfaces (GPIOs) statt. Hierbei ist für das Lichtwecker-Projekt besonders wichtig, dass der Mikrocontroller über viele Schnittstellen und GPIOs verfügt. Somit ist eine einfache Erweiterbarkeit der Funktionalität gewährleistet. Des Weiteren ist ein populäres Modell, bei dem die Entwicklung mit der IDE von Arduino erfolgt, zu wählen, um möglichst viele bereits vorhandene Softwarebibliotheken zu nutzen und auf bekanntes Wissen zurückzugreifen.

Momentan sind zwei Anbieter von Mikrocontrollern für den privaten Gebrauch sehr beliebt: zum einen die Mikrocontroller der Firma Arduino und zum anderen der Firma Espressif Systems.

4.2.1. Arduino Mikrocontroller

Arduino bietet eine Vielzahl von Mikrocontrollern an, die in die Kategorien Einstiegslevel, erweiterte Funktionen und Internet of Things (IoT) gegliedert sind. Aufgrund der vielen verfügbaren Schnittstellen und GPIOs kommen besonders die Mikrocontroller DUE und MEGA, die auch in Abbildung 4.1 und Abbildung 4.2 zu sehen sind, in die nähere Auswahl. Jedoch verfügen diese über kein Modul zur Verbindung mit einem WLAN oder Local Area Network (LAN), somit ist für Netzwerkfunktionen ein zusätzliches Modul notwendig.[9]

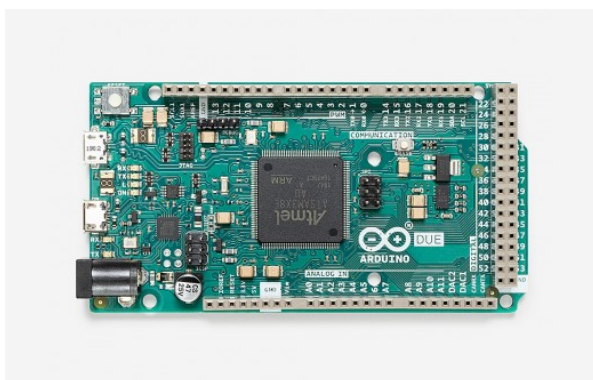


Abbildung 4.1.: Arduino DUE [10]

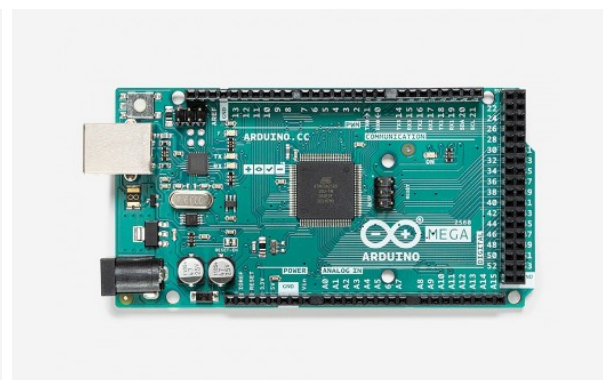


Abbildung 4.2.: Arduino MEGA [11]

4.2.2. Espressif Systems Mikrocontroller

Espressif Systems hingegen hat hauptsächlich den Mikrocontroller ESP32, in Abbildung 4.3 zu sehen, im Portfolio. Dieser ist Nachfolger des bekannten IoT Mikrocontrollers ESP8266. Der ESP32 verfügt über einen Dual-Core Prozessor mit einem 32 Bit Befehlsregister. Zudem ist eine Verbindung zu einem WLAN nach dem IEEE-Standard 802.11 b/g/n möglich. Der Mikrocontroller verfügt über eine Bluetooth 4.2 und Bluetooth Low Energy (BLE) Schnittstelle. Das Layout der Pins hat 34 programmierbare GPIOs, wobei Pulsweitenmodulation (PWM) an bis zu 16 Kanälen möglich ist. Des Weiteren ist der ESP32 mit mehreren Universal Asynchronous Receiver Transmitter (UART) Schnittstellen ausgestattet und unterstützt unter anderem die Bussysteme I2C, Inter-Integrated Circuit Sound (I2S) und Serial Peripheral Interface (SPI). Die Softwareentwicklung ist mit Assembler Programmierung, C-Programmierung in einer beliebigen integrierte Entwicklungsumgebung (IDE) oder mit Hilfe der Arduino IDE und der an C++ angelehnten Programmiersprache umsetzbar. Daraus folgt eine sehr gute Erweiterbarkeit bei Verwendung des ESP32, wobei es auch in der Entwicklung kaum Einschränkungen aufgrund der Hardware und Rechenleistung gibt. Auf Basis dessen fällt die Wahl des Mikrocontrollers auf den ESP32 in der Ausführung ESP32 NodeMCU von AZ-Delivery. [12, S. 8 ff.], [13, S. 5]



Abbildung 4.3.: ESP32 in der Ausführung ESP32 NodeMCU von AZ-Delivery [14]

4.3. Zeit und Zeitsynchronisation

„In den Naturwissenschaften ist die Zeit eine der Basisgrößen des Internationalen Einheitensystems [...]“ [15] Zeit ist eine linear monoton zunehmende Größe, die mittels einer Uhr messbar ist. Eine Uhr kann zum Beispiel (z.B.) ein mechanisches Pendel oder ein Quarz mit Zähler sein. [16, S. 9 ff.]

4.3.1. Zeitstandards

Grundlegend gibt es verschiedene Zeitstandards. Die Bekanntesten sind Temps Atomique International (TAI), Coordinated Universal Time (UTC) und speziell im mitteleuropäischen Raum die mitteleuropäische Zeit (MEZ) und mitteleuropäische Sommerzeit (MESZ). Die TAI ist die internationale Atomzeit. Diese verhält sich chronoskopisch, das bedeutet die TAI ist stetig fortlaufend und hat keine Diskontinuitäten. Die UTC verwendet als Grundlage die TAI und gibt die Uhrzeit nach Erdrotation und Sonne an. Dieser Zeitstandard verhält sich diskontinuierlich aufgrund der Schaltsekunde und enthält somit Sprünge von einer Sekunde. Die MEZ und MESZ leiten sich von der UTC ab und ist die im mitteleuropäischen Raum gebräuchliche Angabe der Uhrzeit. Die Differenz der im Winterhalbjahr verwendeten MEZ beträgt zur UTC +1 Stunde. Hingegen die Differenz der im Sommerhalbjahr verwendeten MESZ beträgt zur UTC +2 Stunden. Für die Entwicklung des Lichtweckers sind sowohl MEZ, als auch MESZ relevant. [16, S. 9 ff.]

4.3.2. Zeitsynchronisationsmechanismen

Zu den Zeitsynchronisationsmechanismen zwischen Computern gehören unter anderem der Berkeley-Algorithmus und das Network Time Protocol (NTP). Bei der Synchronisation nach dem Berkeley-Algorithmus synchronisieren sich mehrere gleichwertige Uhren ermittelt durch einen Zeitserver. Der Zeitserver übermittelt seine Zeit periodisch an die Clients, wonach diese die Differenz zwischen der eigenen Zeit und der Zeitserver-Zeit errechnen und zurück an den Zeitserver senden. Anschließend bildet der Zeitserver den Mittelwert aller Differenzen, die eigene mit einbezogen, und daraus die Korrekturzeit für jeden Client und sich selbst. Daraufhin findet die Übermittlung der clientspezifischen Korrekturen statt. [16, S. 9 ff.]

Über das NTP können einzelne Clients die genaue Uhrzeit von einem Zeitserver erfragen. Dabei ist die Laufzeit der Datenpakete, auf Basis einer Laufzeitmessung, mitberücksichtigt [17]. Für die Zeitsynchronisation über das NTP sind bei einem ESP32 verschiedene Softwarebibliotheken nutzbar. Zu den gängigen Bibliotheken zählen [18] und [19]. Die Softwarebibliothek [18], herausgegeben von Arduino, ist eine sehr simpel gestaltete Bibliothek. Bei dieser ist die Zeit direkt formatiert ausgebenbar, jedoch umfasst [18] nur die notwendigsten Methoden zur Synchronisation und Anzeige der Uhrzeit. Die Softwarebibliothek [19] hingegen verfügt, neben den in [18] enthaltenen Methoden, über eine Vielzahl

von zusätzlichen Methoden zur Zeitsynchronisation und sonstigen Verarbeitung von Zeit. Allerdings ist die Synchronisation über NTP weniger einfach gestaltet. Bei komplexeren Anwendungsfällen ist [19] die bessere Softwarebibliothek zur Zeitsynchronisation und Verarbeitung von Zeit.

4.4. Display

Wie in Kapitel 2 beschrieben, besteht eine Anforderung des Lichtweckers darin, die aktuelle Uhrzeit und das Datum auf einem Display anzuzeigen. Zudem ist eine Benutzeroberfläche notwendig, um verschiedene Parameter einzustellen. Die meistverwendeten Displayarten bei Projekten mit Mikrocontrollern sind LCDs und OLED Displays, wie in Abbildung 4.4 und Abbildung 4.5 abgebildet. Beide Displayarten gibt es in verschiedenen Ausführungen bezüglich (bzgl.) Größe, Farbumfang und Möglichkeiten der Ansteuerung.



Abbildung 4.4.: 0,96 Zoll OLED I2C Display 128x64 Pixel [20]

Die Hardwareauswahl aus [4] legt den Fokus auf die Ansteuerung über I2C [8], um die Anzahl der verwendeten GPIOs möglichst gering zu halten. Somit sind nur zwei Datenleitungen beziehungsweise (bzw.) GPIOs am Mikrocontroller zur Ansteuerung des Displays notwendig. Für die softwareseitige Ansteuerung über I2C ist sowohl für OLED Displays [21], als auch für LCDs [22] jeweils eine bekannte Softwarebibliothek verfügbar. Grundsätzlich besteht die Anforderung an ein Display und die dazugehörige Softwarebibliothek, Zeichen in Form von Buchstaben und Zahlen anzuzeigen, um die Uhrzeit, das Datum und ein Menü darzustellen. Beide Softwarebibliotheken, [21] und [22], stellen Methoden zur Verfügung, dies umzusetzen. Um die von der jeweiligen Softwarebibliothek bereitgestellten Methoden zu nutzen, ist die Deklaration eines Objektes des Displaytyps notwendig. Anschließend sind die Methoden mit dem Objekt nutzbar.

Der Vorteil der Softwarebibliothek für ein OLED Display nach [21] ist, dass sowohl Grafiken, als auch Zeichen darstellbar sind. Hiermit lässt sich eine optisch ansprechende und intuitive Benutzeroberfläche gestalten.

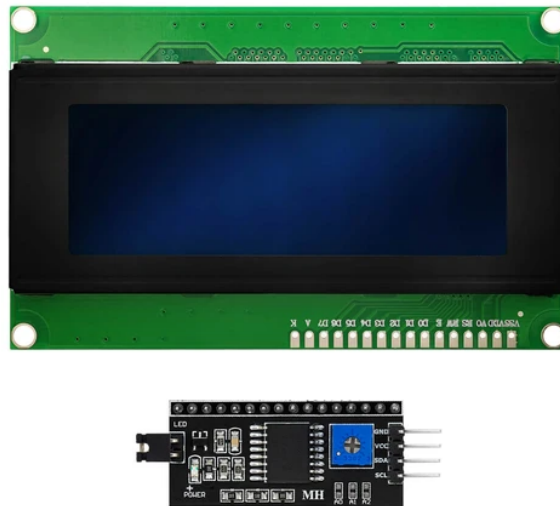


Abbildung 4.5.: HD44780 2004 LCD Bundle 4x20 Zeichen mit I2C Schnittstelle [23]

Die Softwarebibliothek für ein LCD nach [22] hingegen, bietet nur die Möglichkeit zur Darstellung von Zeichen. Dies ist darin begründet, dass es sich um ein 4x20 Zeichen Segment-LCD handelt. Somit ist es, im Gegensatz zum OLED Display, ein in 80 Segmente unterteiltes Display, das pro Segment ein Zeichen anzeigen kann.

Grundsätzlich unterscheiden sich die beiden Displayarten, OLED Display und LCD, bei der softwareseitigen Ansteuerung über I2C, unter Verwendung der jeweiligen Bibliotheken [21] bzw. [22], kaum. Wenn eine grafisch ansprechende Gestaltung gewünscht ist, bietet ein OLED Display jedoch, im Gegensatz zu einem LCD mit Segmenten, deutlich mehr und einfachere Möglichkeiten bei der Umsetzung.

4.5. Beleuchtung

Grundlegender Bestandteil eines Lichtweckers ist ein Leuchtmittel, um die Lichtweckfunktion zu realisieren. Hierbei geht aus den Anforderungen von Kapitel 2 hervor, dass ein Leuchtmittel mit einstellbarer Helligkeit und Farbe zu wählen ist. Somit ist der Wecker auch als Tischlampe oder Nachtlcht einsetzbar. Auf Basis der Anforderungen an

ein Leuchtmittel bieten RGB LEDs, wie in Abbildung 4.6 dargestellt, die besten Eigenschaften, da diese sowohl in Helligkeit, als auch Farbe in einem sehr breiten Spektrum einstellbar sind. RGB LEDs, die, wie in Abbildung 4.6 gezeigt, gebaut sind, verfügen über vier Pins. Drei der Pins dienen zur Ansteuerung über jeweils ein PWM Signal. Somit lassen sich sowohl Helligkeit, als auch Farbe variabel einstellen. [24]

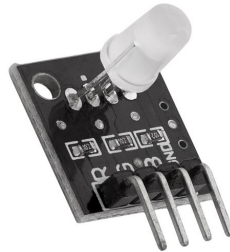


Abbildung 4.6.: LED RGB Modul [25]

Eine Alternative zu einfachen RGB LEDs sind sogenannte Neopixel, wie in Abbildung 4.7 und Abbildung 4.8 gezeigt. Dabei erfolgt die Ansteuerung über eine Datenleitung vom Mikrocontroller. Die Neopixel selbst sind mit einem integrierten Baustein ausgestattet, sodass dieser das Signal vom Mikrocontroller weiterverarbeitet. Beim Zusammenschalten mehrerer Neopixel ergibt sich der enorme Vorteil, dass über eine Datenleitung alle Neopixel individuell ansteuerbar sind. Somit lässt sich die Beleuchtung umfangreicher einstellen. Die Softwarebibliothek nach [26] ist eine der Bekanntesten zur Ansteuerung von Neopixeln. Diese umfasst Methoden, um Farbe und Helligkeit der einzelnen Neopixel einzustellen, wobei lediglich die Farbwerte für Rot, Grün und Blau anzugeben sind. [26], [27], [28]

Beide Umsetzungen, RGB LEDs und Neopixel bieten die Möglichkeit Helligkeit und Farbe einzustellen, wie in Kapitel 2 gefordert. Neopixel sind für den gewünschten Verwendungszweck deutlich geeigneter, da die Spannungsversorgung und Ansteuerung nahezu beliebig vieler Neopixel über drei Leitungen möglich ist. Hinzu kommt, dass diese einzeln ansteuerbar und somit auch einzeln in Helligkeit und Farbe einstellbar sind. Dadurch ist das Projekt zukünftig gut erweiterbar, um aufwendigere bzw. elegantere Wecklichter zu gestalten.

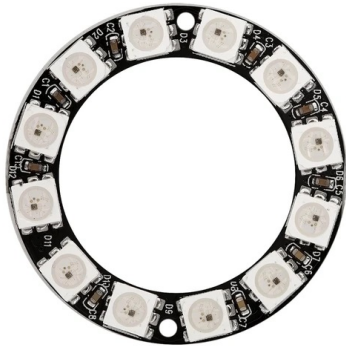


Abbildung 4.7.: Neopixel Ring [29]

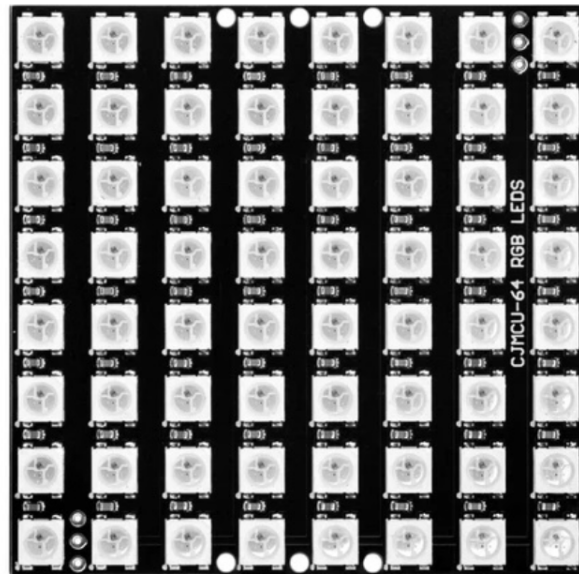


Abbildung 4.8.: Neopixel Matrix [30]

4.6. Bedienelemente

Da ein Lichtwecker auch die Eingabe durch einen Benutzer benötigt, sind Bedienelemente erforderlich. Hier kommen typischerweise Drehwinkelgeber, wie in Abbildung 4.10 zu sehen, oder Taster, wie in Abbildung 4.9 dargestellt, zum Einsatz.

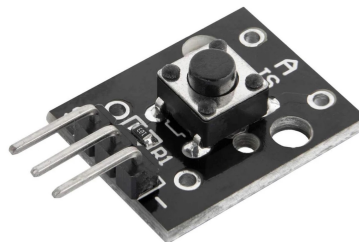


Abbildung 4.9.: Taster [31]

Wie in Abbildung 4.11 gezeigt, verfügen Drehwinkelgeber über eine feste Anzahl von Positionen pro Umdrehung und drei Pins zur Abtastung. Somit ist es möglich die Drehrichtung und Anzahl der gedrehten Positionen zu bestimmen. Für die Bestimmung dieser Parameter ist eine softwareseitige Implementierung notwendig. Taster hingegen, wie in Abbildung 4.9 dargestellt, verfügen über die zwei Zustände: gedrückt und nicht gedrückt. [32], [33]



Abbildung 4.10.: Drehwinkelgeber [34]

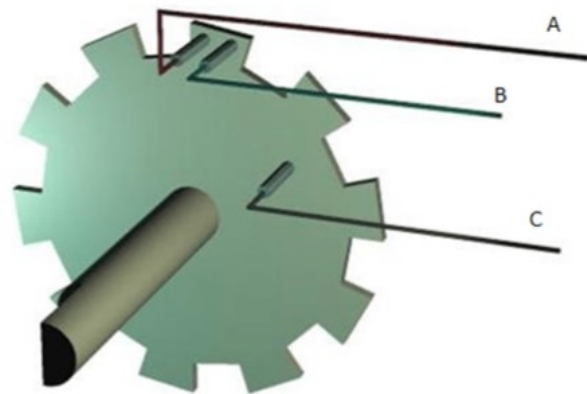


Abbildung 4.11.: Abtastung Drehwinkelgeber [32, S. 3]

Beide Bedienelemente, Drehwinkelgeber und Taster, sind im Grunde mechanische Kontakte. „[Diese] haben den Nachteil, dass sie durch ihren mechanischen Aufbau prellen. Werden sie betätigt, wird durch das Kippen eine Kraft auf den Kontakt ausgelöst, die ihn im Submillimeterbereich mehrmals schließen und öffnen lässt. Das Prellen entsteht genau an diesem Kontakt, zu dem der Schalter bewegt wird.“[35] Vereinfacht ist der Vorgang des Prellens in Abbildung 4.12 dargestellt. Prellen ist grundsätzlich zu vermeiden bzw. zu beheben durch hardwareseitiges oder softwareseitiges Entprellen. Per Software ist dies mit Hilfe des Warteschleifen-Verfahren realisierbar. Bei dieser Art des Entprellens ist die vergangene Zeit, seit des letzten Signal- bzw. Flankenwechsels zu prüfen. Das Warteschleifenverfahren stellt die Erfassung nur gewünschter Signalpegel sicher. [35], [36]



Abbildung 4.12.: Prellen eines mechanischen Kontaktes [37]

Im Regelfall sind Drehwinkelgeber und Taster mit einem Pull-Up oder Pull-Down Widerstand, wie in Abbildung 4.13 und Abbildung 4.14 schematisch dargestellt, geschaltet. Somit ist die Schaltung vor Störsignalen geschützt. Dies ist sowohl hardwareseitig, als auch softwareseitig (beim ESP32) umsetzbar. Bei der Schaltung mit Pull-Up Widerstand,

wie in Abbildung 4.13, ist der mechanische Kontakt „Active-Low“, sodass ein Tastendruck einen LOW-Pegel liefert. Dies geschieht, da der Pull-Up Widerstand, die Spannung auf die Betriebsspannung, zieht. Der mechanische Kontakt hingegen, ist, wie in Abbildung 4.14, mit dem Pull-Down Widerstand als „Active-High“ geschaltet. In dieser Schaltung gibt ein Tastendruck einen HIGH-Pegel, da der Pull-Down Widerstand die Spannung auf Masse zieht. [8]

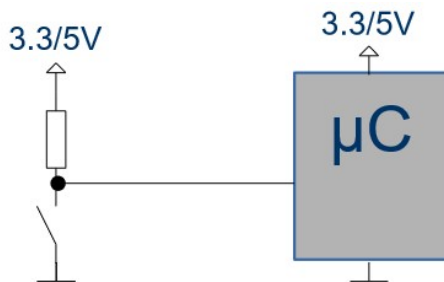


Abbildung 4.13.: Taster mit Pull-Up Widerstand [8]

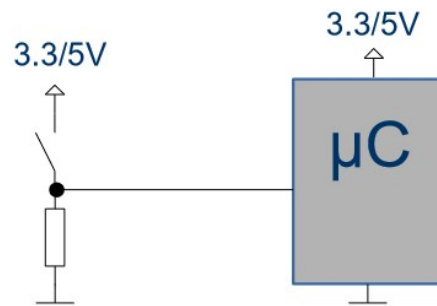


Abbildung 4.14.: Taster mit Pull-Down Widerstand [8]

4.7. Tonausgabe

Um das Aufwachen zur gewünschten Uhrzeit zu gewährleisten, ist eine zusätzliche akustische Weckfunktion notwendig. Hierfür ist, wie auch in Kapitel 2 beschrieben, eine Tonausgabe umzusetzen. Dabei bietet sich sowohl ein aktiver, als auch ein passiver piezoelektrischer Buzzer, wie in Abbildung 4.15 und Abbildung 4.16 dargestellt, oder ein MP3 Player Modul, wie in Abbildung 4.17 zu sehen, für die Umsetzung an.

Der aktive piezoelektrische Buzzer, wie in Abbildung 4.15 zu sehen, kann lediglich einen Ton mit einer Frequenz von circa 2,5 kHz ausgeben. Somit ist die softwareseitige Ansteuerung sehr einfach in der Handhabung. Sobald ein High-Signal am Modul anliegt, erzeugt dieses einen Ton. Daraus folgt, dass lediglich ein dauerhafter oder piepender akustischer Alarmton realisierbar ist. [38]

Der passive piezoelektrische Buzzer hingegen, wie in Abbildung 4.16 dargestellt, kann Töne im Bereich von circa 1,5 kHz bis 2,5 kHz, je nach Frequenz des angelegten Signals, erzeugen. Die softwareseitige Ansteuerung erfolgt mit Hilfe eines PWM-Signals, um die gewünschte Signalfrequenz auszugeben. Somit sind zusätzlich, zu den Funktionalitäten

eines aktiven Buzzers, Melodien als Alarmton realisierbar. Dies erfordert, je nach Komplexität und Länge der Melodie, einen unterschiedlichen Arbeitsaufwand bei der Umsetzung. [39]



Abbildung 4.15.: Buzzer Modul aktiv [40]

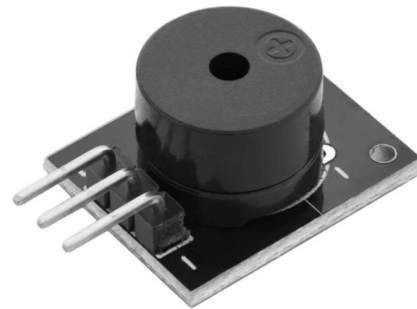


Abbildung 4.16.: Buzzer Modul passiv [41]

Die modularste und komfortabelste Lösung für die Tonausgabe bietet ein MP3 Player Modul, wie in Abbildung 4.17 gezeigt, in Verbindung mit einem oder zwei Lautsprechern. Im folgenden ist nur das MP3 Player Modul thematisiert, da nur dieses softwareseitig relevant ist (genauere Details zur Hardware in Verbindung mit Lautsprechern sind in [4] zu finden). Das Modul ist seriell ansteuerbar und verfügt über eine integrierte MP3 Dekodierung. Des Weiteren unterstützt das MP3 Player Modul Micro Secure Digital Memory Cards (microSDs), sodass MP3 Dateien von dieser lesbar sind. Die Ansteuerung erfolgt, wie in [42, S. 5–13] beschrieben, über eine serielle Schnittstelle. Dabei sind die Musikwiedergabe und Eigenschaften der Wiedergabe, wie z.B. Lautstärke, steuerbar. Um die Ansteuerung einfacher zu gestalten, ist der Einsatz der Softwarebibliothek nach [43] sinnvoll. Somit sind nach der Deklaration eines Objektes der Bibliothek, auch die Methoden dieser nutzbar. Hierdurch ist es nicht erforderlich Funktionen zum Senden und Empfangen von Daten vom MP3 Player Modul zu implementieren. [42], [43], [44]

Die einfachste Umsetzung eines Alarmtons ist mit einem aktiven piezoelektrischen Buzzer möglich. Jedoch bietet dieser nur eingeschränkte Möglichkeiten bei der Tonausgabe, wodurch der eigentliche Nutzen eines Lichtweckers, das entspannte Aufwachen, hinfällig ist. Eine bessere Möglichkeit stellt der passive piezoelektrische Buzzer dar, da mit diesem auch die Wiedergabe von Melodien realisierbar ist. Jedoch ist der Aufwand größer, je komplexer die Melodie ist. Zudem kann der Nutzer, bei dieser Umsetzung, im besten Fall nur eine vorgefertigte Melodie auswählen. Bei der Umsetzung mit einem MP3 Player Modul hingegen, kann der Benutzer die gewünschten Melodien und Lieder selbständig auf die

microSD kopieren. Dadurch entsteht ein hoher Personalisierungsgrad beim Alarmton und die Musikwiedergabe, als Zusatzfunktion, ist umsetzbar. Somit ist die Umsetzung der Tonausgabe mit einem MP3 Player Modul in Kombination mit der Softwarebibliothek nach [43] zu priorisieren.

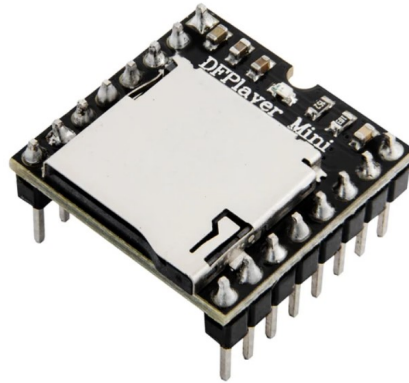


Abbildung 4.17.: MP3 Player Modul [45]

4.8. Echtzeit

„Ein Echtzeitsystem ist ein System, bei dem der Zeitpunkt, zu dem Ausgaben erzeugt werden, bedeutend ist. [...] Die Verzögerung zwischen der Zeit der Eingabe und der Zeit der Ausgabe muss ausreichend klein für eine akzeptable ‚Rechtzeitigkeit‘ (engl. *timeliness*) sein.“ [5, S. 39 f.]

Echtzeit lässt sich in harte, feste und weiche Echtzeitbedingungen unterscheiden. Bei der harten Echtzeit ist das Überschreiten einer Deadline unter keinen Umständen zulässig, da sonst katastrophale Folgen oder sehr hohe Kosten zu befürchten sind. Feste Echtzeit bedeutet, dass eine Überschreitung der Deadline das Ergebnis nutzlos macht. Jedoch kommt es, im Gegensatz zur harten Echtzeit, zu keinem unmittelbaren Schaden. Bei weichen Echtzeitbedingungen sind Verzögerungen bzw. Deadline-Überschreitungen tolerierbar, wobei die Zeitbedingungen als Richtwerte, mit einer Toleranz bei Abweichungen, dienen. [5, S. 40], [46, S. 321 f.]

„Die Bewertung der Überschreitung von Zeitbedingungen kann auch durch eine Wertfunktion (Time Utility Function) erfolgen. Diese Funktion gibt den Wert einer Aktion in Abhängigkeit der Zeit an.“ [46, S. 322] Dieser Zusammenhang bei harter, fester und weicher Echtzeit ist in Abbildung 4.18 dargestellt. Dabei bedeutet ein positiver Wert, dass

eine Aktion von Nutzen ist. Beim Erreichen des Wertes 0 ist die Aktion nutzlos und führt bei negativen Werten sogar zu Schaden. [46, S. 322]

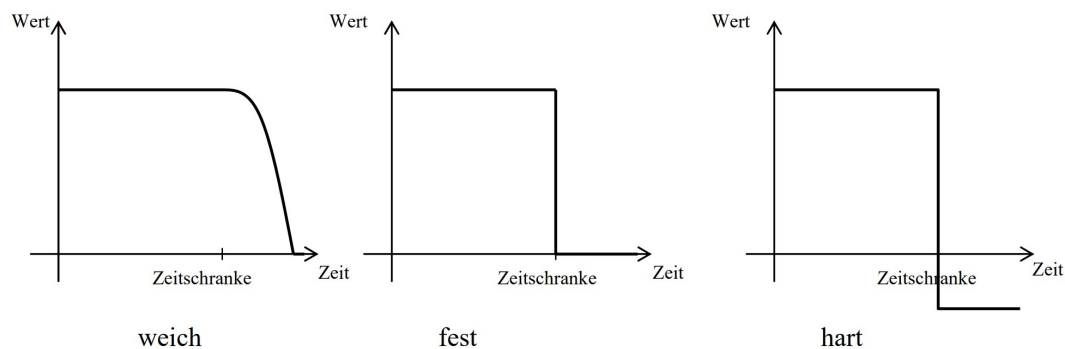


Abbildung 4.18.: Wertfunktionen zur Bewertung von Zeitbedingungen [46, S. 322]

Im Fall des Lichtweckers handelt es sich bei der Anzeige der Uhrzeit und der Weckfunktion um feste Echtzeit. Die Überschreitung der Zeitbedingungen führt dazu, dass die Anzeige der Uhrzeit bzw. die Weckfunktionalität den Nutzen verlieren. Dabei entsteht kein unmittelbarer Schaden. Sowohl die Anzeige der Uhrzeit, als auch die Weckfunktionalität haben dabei absolute Zeitbedingungen. Da die Anzeige der Uhrzeit auch die aktuelle Sekunde umfasst, ist dabei eine sekundengenaue Anzeige notwendig. Bei der Weckfunktion hingegen, ist die gewünschte Weckzeit typischerweise nur auf eine Minute genau einstellbar. Hieraus ergibt sich die Anforderung einer minutengenauen Umsetzung. [46, S. 320]

4.9. Vergleichbare Projekte

Diese Studienarbeit ist nicht das erste Projekt zur Realisierung eines Lichtweckers. Ein solcher ist bereits teilweise in [47], einer vorangegangene Studienarbeit, umgesetzt. Zudem sind im Internet ähnliche Projekte auffindbar, wie z.B. in [48], [49] und [50].

Die Projekte nutzen unterschiedliche Arduino Mikrocontroller, den UNO [49] und MEGA [48], bzw. einen ESP8266 der Firma Espressif Systems [50]. Somit ist es nur bei dem Projekt aus [50] möglich, eine Internetverbindung, ohne zusätzliches Modul, herzustellen. Daraus ergibt sich auch, dass die Projekte aus [48] und [49] eine Real-Time Clock (RTC) zur Zeitsynchronisation nutzen. Das Projekt aus [50] hingegen kann, aufgrund der Internetfähigkeit des Mikrocontrollers, die Zeit über das NTP synchronisieren.

Zur Umsetzung der Lichtweckfunktionalität nutzen die genannten Projekte einfarbige bzw. RGB Neopixel mit gleicher Ansteuerung, wie zuvor in Abschnitt 4.5 beschrieben. Einzig das Projekt aus [47] beinhaltet als Projektbestandteil die zusätzliche Nutzung der Neopixel für normale Beleuchtung. Die anderen Projekte hingegen, nutzen die Lichtquelle nur für die Lichtweckfunktion. Alle vier Projekte legen dabei keinen Fokus auf einen angenehmen Weckton, der typischerweise, trotz Lichtweckfunktionalität, erforderlich ist. Das Projekt in [47] verfügt lediglich über einen Signalton zum Wecken. Die Projekte aus [48], [49] und [50] hingegen, verzichten komplett auf eine akustische Weckfunktion.

Die Anzeige, Bedienung und Einstellung erfolgt bei den Projekten aus [48] und [47] über ein LCD bzw. E-Paper Display. Für die Eingabe des Benutzers kommen Taster und Drehwinkelgeber zum Einsatz. Das Projekt aus [50] hingegen verzichtet komplett auf eine Anzeige über ein Display oder die Bedienung mit physischen Bedienelementen. Hier erfolgt die Anzeige und Bedienung über ein Webinterface.

Im Allgemeinen sind, bei den hier erwähnten Projekten, die Einstellmöglichkeiten für den Benutzer stark eingeschränkt. Dabei ist es z.B. teilweise nicht möglich mehrere Weckzeiten oder die Wochentage, an den eine Wiederholung stattfinden soll, einzustellen. [47], [48], [49], [50]

5. Lösungsbeschreibung und Umsetzung

Dieses Kapitel der Studienarbeit I umfasst die softwareseitige Lösungsbeschreibung und Umsetzung der „Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion (Software)“. Dabei ist der grundlegende Aufbau des Projektes, sowie die Implementierung der einzelnen Komponenten bzw. Module und der Weckfunktion, thematisiert. Des Weiteren beschreibt dieses Kapitel die Zusammenführung der Einzelkomponenten und der Weckfunktion mit Hilfe einer Menüstruktur. Über diese erfolgt auch die Realisierung der Benutzerinteraktion und -eingabe.

5.1. Verwendete Hardware

Dieser Abschnitt thematisiert die verwendete Hardware. Die softwareseitige Vorbetrachtung zur Auswahl der Hardware ist bereits in Kapitel 4 thematisiert. Genauere Informationen zur Hardwareauswahl sind [4], der hardwareseitigen Studienarbeit, zu entnehmen.

Für die Umsetzung des Projektes kommt der Mikrocontroller ESP32, in der Ausführung NodeMCU, aufgrund der zuvor in Unterabschnitt 4.2.2 beschriebenen Eigenschaften, zum Einsatz. Die Anzeige der Uhrzeit und der Einstellmöglichkeiten sind über das in Abschnitt 4.4 beschriebene LCD realisiert. Zur Bedienung sind fünf Taster, wie in Abschnitt 4.6 aufgeführt, eingesetzt. Für die Realisierung der Lichtweckfunktion kommt die aus Abschnitt 4.5 thematisierte Neopixel Matrix, bestehend aus 64 Neopixeln, zum Einsatz. Die akustische Weckfunktion ist mit einem MP3 Player Modul, wie in Abschnitt 4.7 beschrieben, umgesetzt.

5.2. Aufbau der Software

Die Softwareentwicklung des Lichtweckers erfolgt mit der Arduino IDE. Um diese zur Entwicklung von Programmen für den ESP32 zu nutzen, ist in den Voreinstellungen unter „Zusätzliche Boardverwalter-URLs“ der folgende Link einzufügen: https://dl.espressif.com/dl/package_esp32_index.json. Des Weiteren ist im „Boardverwalter“ die Installation des Paketes „esp32“ von Espressif Systems durchzuführen. Nach Abschluss der Installation ist die Softwareentwicklung mit Hilfe der Arduino IDE möglich. [51]

„Sketch“ heißt ein in der Arduino IDE entwickeltes Programm [52]. Ein Sketch kann sich über mehrere .ino-Dateien erstrecken bzw. ist es auch möglich Header- und C++-Dateien einzubinden. Der Sketch des Lichtweckers umfasst die folgenden Dateien:

- 2021-03-27_V28.ino (Listing A.1)
- alarm.ino (Listing A.2)
- backEvent.ino (Listing A.3)
- light.ino (Listing A.4)
- ntp.ino (Listing A.5)
- other.ino (Listing A.6)
- selectEvent.ino (Listing A.7)
- setup.ino (Listing A.8)
- writeDisplay.ino (Listing A.9)

Die wichtigste Datei des Sketches ist „2021-03-27_V28.ino“. Diese umfasst alle Bezeichner (`#define`), die Einbindung der Softwarebibliotheken, die globalen Variablen, Datenstrukturdefinitionen, die ISRs, die `setup()`-Funktion und die `loop()`-Funktion. Zu Beginn des Sketches, in Listing A.1 (Zeile 1-8), kommt mehrfach `#include <libraryFile.h>` zum Einsatz, um externe Softwarebibliotheken aufzunehmen [53]. „Wenn die Syntax der spitzen Klammern verwendet wird, werden die Bibliothekspfade nach der Datei durchsucht. [...] Wenn [hingegen] die Syntax in doppelten Anführungszeichen verwendet wird, wird der Ordner der Datei mit der Direktive `#include` nach der angegebenen Datei durchsucht. Anschließend wird in den Bibliothekspfaden gesucht, wenn sie nicht im lokalen Pfad gefunden wurden.“[53] Die in Listing A.1 verwendeten Bibliotheken sind zuvor über den „Bibliotheksverwalter“ zu downloaden. Mit Hilfe der Bezeichner (`#define`), welche sich

in Listing A.1 (Zeile 10-20) befinden, ist es möglich, konstanten Werten einen Namen vor dem Kompilieren zuzuweisen [54]. Damit ist eine bessere Lesbarkeit des Quellcodes gewährleistet.

Hauptbestandteil des Sketches sind die Funktionen `setup()` und `loop()`. Der Aufruf der Funktion `setup()`, in Listing A.1 (Zeile 142-159), findet immer nur ein einziges Mal statt. Dies geschieht entweder beim Starten bzw. nach dem Resetten des ESP32 [55]. Die Funktion dient dabei zum Initialisieren von z.B. Pinmodi oder Bibliotheken [55]. Die Funktion `loop()`, in Listing A.1 (Zeile 162-267), ist hingegen eine Endlosschleife [56]. Somit wird die Funktion nach jedem Durchlauf erneut aufgerufen und das Board ist aktiv steuerbar [56]. Von der `loop()`-Funktion ausgehend, erfolgt, wie in Abbildung 5.1 dargestellt, der zyklische bzw. ereignisbasierte Aufruf anderer Funktionen. Des Weiteren enthält der Sketch fünf ISRs, die den Hauptablauf, beim Auslösen des jeweiligen Hardwareinterrupts, unterbrechen.

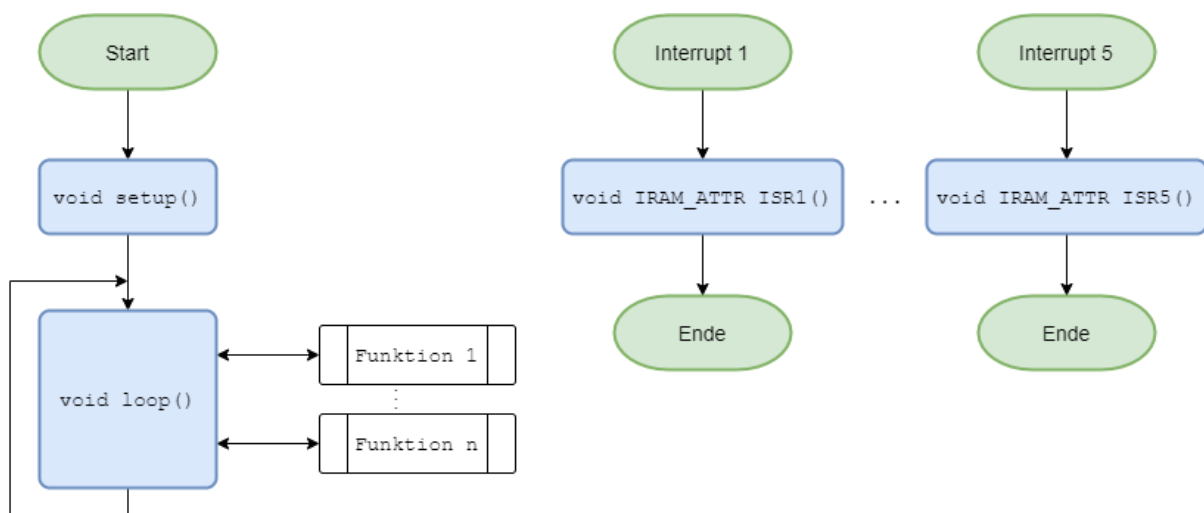


Abbildung 5.1.: Ablaufdiagramm zum Aufbau der Software

5.3. Implementierung der einzelnen Komponenten

Die nachfolgenden Abschnitte thematisieren die Implementierung der einzelnen Komponenten bzw. Module. Dies umfasst die Implementierung der Zeit und Zeitsynchronisation über das NTP, des WLAN Managers, der Bedienelemente, des LCD, des LED Panels und des MP3 Player Moduls.

5.3.1. Zeit und Zeitsynchronisation

Für die Umsetzung der Zeit und Zeitsynchronisation über das NTP kommt, die zuvor in Unterabschnitt 4.3.2 beschriebene Softwarebibliothek „Time“ [19] zum Einsatz. Die Bibliothek enthält eine Vielzahl von Funktionen zur Verarbeitung von Zeit. Zur Zeitsynchronisation über das NTP stellt die Softwarebibliothek die Funktionen `getNtpTime()`, wie auch in Listing A.5 (Zeile 3-31) dargestellt, und `sendNTPpacket(IPAddress &address)`, wie in Listing A.5 (Zeile 35-54) gezeigt, zur Verfügung.

Um diese Funktionen einzusetzen, muss eine Netzwerkverbindung bestehen. Dies wird zunächst mit statischem Netzwerknamen und -password, wie in Listing 5.1, gelöst. Die finale Umsetzung der Netzwerkverbindung, mit der Möglichkeit, das Netzwerk zu wechseln, ist im nachfolgenden Unterabschnitt 5.3.2, beschrieben. Zudem ist eine User Datagram Protocol (UDP)-Verbindung, zum Senden und Empfangen von Paketen, notwendig. Die Funktion `udp()`, in Listing A.8 (Zeile 27-32), setzt dies um.

Listing 5.1: Statische WLAN-Verbindung

```
1 const char* ssid          = "SSID";           //network name
2 const char* password     = "EnterYourPassword"; //network password
```

Die Funktion `ntp()`, in Listing A.8 (Zeile 100-105), weist zum einen die Methode zur Zeitsynchronisation zu. Zum andern wird die Zeitzone, sowie das Synchronisationsintervall festgelegt. Anschließend findet die Synchronisation, über das NTP, automatisch im definierten Intervall statt. Für die manuelle Einstellung von Uhrzeit und Datum, in Listing A.7 (Zeile 461-503) zu sehen, kommt die Funktion `setTime(int hr, int min, int sec, int day, int month, int yr)` der Bibliothek „Time“ zum Einsatz. Zudem ist die Zeitzone bzw. Sommer-/Winterzeit, durch Änderung der Variable `int timeZone`, wie in Listing A.7 (Zeile 448-454), einstellbar.

Des Weiteren sind Funktionen der Softwarebibliothek in `digitalClockDisplay()` aus Listing A.9 (Zeile 171-191), für die Anzeige der aktuellen Uhrzeit und des aktuellen Datums, genutzt.

5.3.2. WLAN Manager

Wie zuvor in Unterabschnitt 5.3.1 beschrieben, wird die Netzwerkverbindung zunächst statisch umgesetzt. Die statische Umsetzung ist nicht benutzerfreundlich, da die Netzwerkverbindung nur in einem Netzwerk mit gleichem Namen und Passwort möglich ist. Für die dynamische Umsetzung kommt die Bibliothek „WiFiManager“ [57] zum Einsatz. Die Verbindung zu einem WLAN ist in der Funktion `wlan()`, in Listing A.8 (Zeile 43-71), realisiert. Mit Hilfe der Methode `autoConnect(char const *apName, char const *apPassword = NULL)`, der Bibliothek, kann sich der ESP32, nach dem Starten, mit einem zuvor gespeicherten WLAN verbinden. Wenn dies nicht möglich ist, wird der ESP32 zu einem Access Point, mit dem an die Methode übergebene Netzwerknamen und -passwort. Im Fall des Lichtweckers hat das Netzwerk, wie in Abbildung 5.2 zu sehen, den Namen „Lichtwecker Setup“ und eine Verbindung ohne Passwort ist möglich.

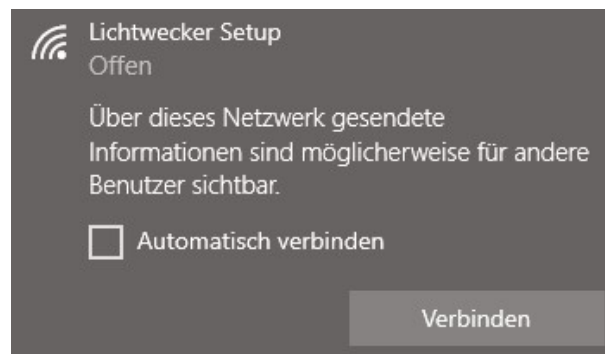


Abbildung 5.2.: Access Point „Lichtwecker Setup“

Des Weiteren startet die Methode einen Webserver und Domain Name System (DNS)-Server, sodass sich, nach dem Verbinden mit dem Netzwerk „Lichtwecker Setup“, das, in Abbildung 5.3 zu sehende Webinterface, öffnet. Hierfür ist jedes WLAN-fähige Endgerät mit Browser einsetzbar. Über das in Abbildung 5.3 dargestellte Webinterface kann der Benutzer nun „Configure WiFi“ auswählen, woraufhin das Webinterface aus Abbildung 5.4 erscheint. Im Webinterface „Configure WiFi“ sind alle WLANs in der Umgebung gelistet, sodass eine neue Verbindung herstellbar ist.

Mit der soeben beschriebenen Funktion lassen sich die Netzwerkeinstellungen nur ändern, wenn sich kein bekanntes WLAN in der Nähe befindet. Die Funktion `wifiManager()`, aus Listing A.8 (Zeile 74-92), ermöglicht es, unter Verwendung der Methode `startConfigPortal(char const *apName, char const *apPassword = NULL)` der Bibliothek „WiFiManager“, das Konfigurationsportal zu starten. Dabei kann der ESP32 auch mit einem bekannten WLAN verbunden sein. Diese Einstellung des WLANs ist auch in die Menüstruktur integrierbar.

Lichtwecker Setup

WiFiManager

Configure WiFi

Info


Exit

Connected to SSID
with IP 192.168.189.21

Abbildung 5.3.: Webinterface:
Lichtwecker
Setup

Martin Router King 

Campofranco 

TP-Link_7384 

SSID

SSID

Password

Save

Refresh

Connected to SSID
with IP 192.168.189.21

Abbildung 5.4.: Webinterface:
Configure Wi-
Fi

5.3.3. Bedienelemente - Taster

Für die Bedienung des Lichtweckers kommen fünf Taster, mit den folgenden Funktionen, zum Einsatz:

- UP
- DOWN
- SELECT
- BACK
- ALARM

Die ersten vier Bezeichnung sind für sich selbst sprechend. Der Taster „ALARM“ dient zum Ausschalten des Alarms.

In der Funktion `input()`, aus Listing A.8 (Zeile 2-8), sind alle fünf Pins, die mit den Tastern verbunden sind, als Input konfiguriert. Zudem sind diese durch den Parameter `INPUT_PULLUP`, wie zuvor in Abschnitt 4.6 thematisiert, mit einem Pull-Up Widerstand geschaltet.

Um einen Tastendruck jederzeit zu erfassen, wird ein Tastendruck als externer Interrupt gehandhabt. Die Konfiguration der externen Interrupts erfolgt in der Funktion `interrupt()`, in Listing A.8 (Zeile 18-24). Da die Taster mit Pull-Up Widerstand geschaltet sind und um den jeweiligen Interrupt zu triggern, ist das Parameter `FALLING` gesetzt. Zudem ist jeweils die ISR definiert, die bei einem Interrupt aufzurufen ist. Die zu jedem Taster zugehörige ISR ist in der nachfolgenden Tabelle aufgelistet:

Tabelle 5.1.: Externe Interrupts und dazugehörige ISRs

Taster	ISR
UP	<code>pressedUP()</code> in Listing A.1 (Zeile 102-109)
DOWN	<code>pressedDOWN()</code> in Listing A.1 (Zeile 92-99)
SELECT	<code>pressedSELECT()</code> in Listing A.1 (Zeile 112-119)
BACK	<code>pressedBACK()</code> in Listing A.1 (Zeile 122-129)
ALARM	<code>pressedALARM()</code> in Listing A.1 (Zeile 132-139)

Um die Taster zu entprellen, kommt in der jeweiligen ISR das Warteschleifenverfahren, wie zuvor in Abschnitt 4.6 beschrieben, zum Einsatz. Dabei wird mit Hilfe der `millis()`-Funktion geprüft, wie viel Zeit seit dem letzten Tastendruck bzw. Pegelwechsel vergangen ist. Wenn die Zeitdifferenz größer 200 Millisekunden ist, handelt es sich definitiv um einen Tastendruck und nicht um das Prellen eines Tasters. Dies ist in Abbildung 5.5 als Ablaufdiagramm dargestellt. Die Verwendung von `millis()` hat gegenüber von `delay(unsigned long ms)` den Vorteil, dass keine Unterbrechung des Programms stattfindet und es somit zu keiner Verzögerung kommt.

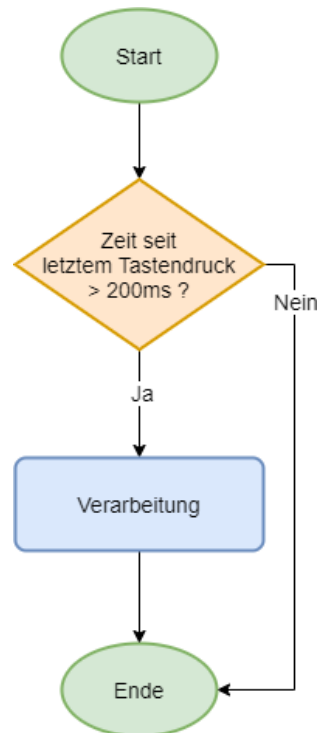


Abbildung 5.5.: Ablaufdiagramm einer ISR

5.3.4. LCD

Für die Anzeige von Uhrzeit und Datum, sowie der Menü- und Konfigurationsoberfläche, kommt, wie bereit in Abschnitt 5.1 thematisiert, ein 4x20 Zeichen LCD mit Ansteuerung über I2C, zu Einsatz. Softwareseitig ist die Ansteuerung mit Hilfe der Bibliothek „LiquidCrystal_I2C“ [22] in Verbindung mit „Wire“ realisiert.

Beim Starten des ESP32 wird das LCD in der Funktion `lcd()`, wie in Listing A.8 (Zeile 35-50), initialisiert. Mit Hilfe der Methoden `setCursor(uint8_t col, uint8_t row)` und `print(data)`, der Bibliothek „LiquidCrystal_I2C“, ist die Anzeige von Daten auf dem LCD, wie in Listing A.9, realisiert. Dies ist beispielhaft in Listing 5.2 gezeigt. Um alle angezeigten Zeichen vom LCD zu entfernen, kommt die Methode `clear()`, wie auch in Listing A.1 (Zeile 165), zur Verwendung.

Listing 5.2: Ausgabe von vier Zeilen Text auf LCD

```
1 display.setCursor(0,0);
2 display.print("Text in Zeile 1");
3 display.setCursor(0,1);
4 display.print("Text in Zeile 2");
```

```
5 display.setCursor(0,2);
6 display.print("Text in Zeile 3");
7 display.setCursor(0,3);
8 display.print("Text in Zeile 4");
```

Über das Menü hat der Benutzer die Möglichkeit die Hintergrundbeleuchtung des LCD ein- bzw. auszuschalten. Dies ist mit den Methoden `noBacklight()` und `backlight()` der Bibliothek „LiquidCrystal_I2C“, in Listing A.7 (Zeile 83-88), realisiert.

5.3.5. Beleuchtung - LED Panel

Als Leuchtmittel, für die Lichtweckfunktion und als Lampe, kommt die zuvor in Abschnitt 4.5 thematisierte, Neopixel Matrix mit 64 Neopixeln zum Einsatz. Die softwareseitige Ansteuerung ist mit der Bibliothek „Adafruit NeoPixel“ [26] umgesetzt. Die Softwarebibliothek stellt die Methode `setPixelColor(uint16_t n, uint32_t c)` bereit, womit sich die Farbe eines einzelnen Neopixel setzen lässt. Mit Hilfe der Methode `show()` sind die gesetzten Farbwerte an die Neopixel zu übertragbar.

Der Aufruf der Funktion `setLight(uint8_t r, uint8_t g, uint8_t b)`, aus Listing A.4 (Zeile 2-5), setzt die Farbwerte aller Neopixel auf die übergebenen Farbwerte und überträgt diese anschließend an die Neopixel. Dies ist in Form eines Ablaufdiagramms in Abbildung 5.6 dargestellt.

Beim Starten des ESP32 liest die Funktion `light()`, in Listing A.8 (Zeile 120-127), die letzten gesetzten Farbwerte aus dem Electrically Erasable Programmable Read-Only Memory (EEPROM). Des Weiteren setzt die Funktion die Farbwerte aller Neopixel und überträgt diese an die Neopixel.

Die Nutzung der Neopixel Matrix, als normal Beleuchtung, ist über ein Menü möglich. Dies ist in Listing A.7 (Zeile 37-57) und Listing A.3 (Zeile 13-21) realisiert. Dabei kann der Benutzer die einzelnen Farbwerte rot, grün und blau in einem Wertebereich von 0 bis 250 einstellen. Zudem lässt sich die Matrix über einen Menüpunkt Ein- und Ausschalten. Beim Verlassen des Menüs wird die Konfiguration der Farbwerte im EEPROM gespeichert. Somit gehen die Konfigurationen, bei einem Neustart des ESP32, nicht verloren.

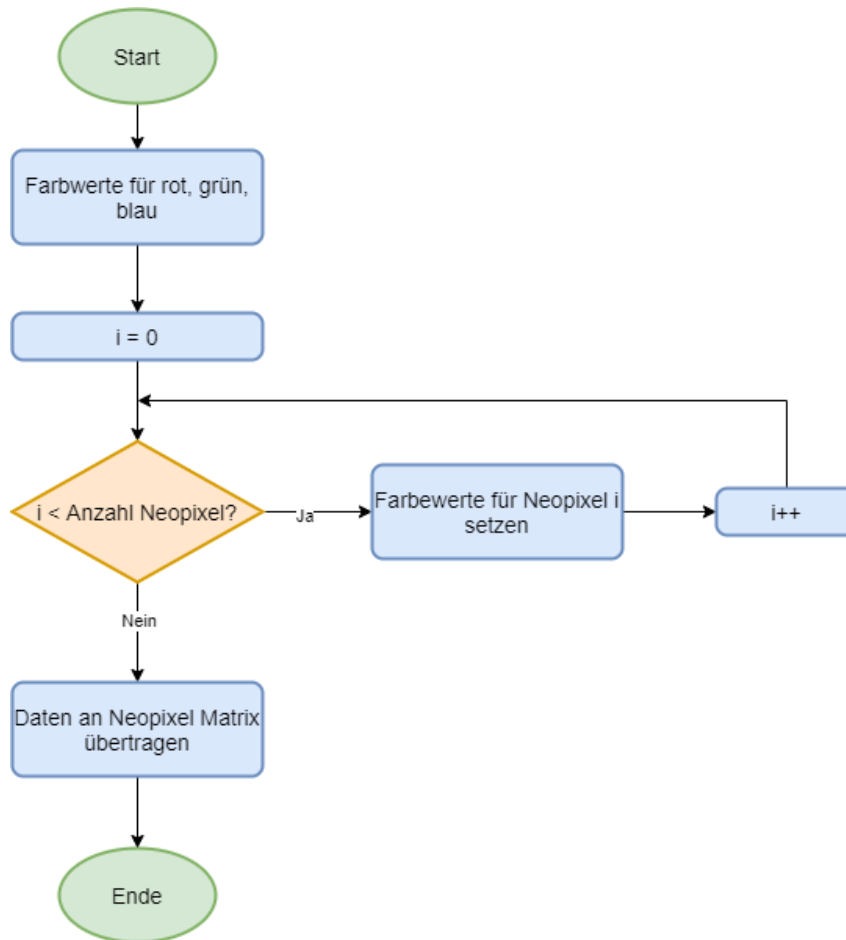


Abbildung 5.6.: Ablaufdiagramm zur Ansteuerung des LED Panels

5.3.6. Tonausgabe - MP3 Player

Die Tonausgabe ist mit einem MP3 Player Modul umgesetzt. Da die Ansteuerung über eine serielle Schnittstelle erfolgt, wird diese beim Start des Mikrocontrollers, in Listing A.1 (Zeile 144), initialisiert. Die Initialisierung des Moduls findet anschließend in der `mp3()`-Funktion, in Listing A.8 (Zeile 108-117), statt. Um die Ansteuerung zu erleichtern, kommt die, zuvor in Abschnitt 4.7 thematisierte, Bibliothek „DFRobotDFPlayerMini“ [43], zum Einsatz.

Grundlegend finden hauptsächlich die Methoden `play(int fileNumber=1)`, `pause()`, `next()` und `volume(uint8_t volume)`, der Softwarebibliothek, Anwendung. Mit Hilfe dieser Methoden ist es möglich, eine spezifische MP3 Datei von der microSD wiederzugeben, die Wiedergabe zu unterbrechen, die nächste MP3 Datei abzuspielen und die Wiedergabelautstärke anzupassen.

Das MP3 Player Modul wird zum einen als akustisches Wecksignal, wie in Listing A.2 (Zeile 35 f.), eingesetzt. Zum anderen ist der Lichtwecker auch, über das Menü, in Listing A.7 (Zeile 60-78) zu sehen, als einfacher MP3 Player nutzbar.

5.4. Weckfunktionen

Die folgenden Abschnitte umfassen die Umsetzung und Implementierung der Weckfunktionen. Dies beinhaltet die Implementierung einer Datenstruktur, um einen Wecker mit erweiterter Funktionalität, zu speichern. Zudem ist die Umsetzung der akustischen Weckfunktion, sowie der Lichtweckfunktion, thematisiert.

5.4.1. Datenstruktur

Grundfunktion des Lichtweckers ist das Wecken. Dabei ist eine programmierbare Wochenendabschaltung, wie in Kapitel 2 definiert, umzusetzen. Des Weiteren sind über das MP3 Player Modul individuelle akustische Signale bzw. Lieder abspielbar. Somit ist eine individuelle Einstellung des akustischen Signals umsetzbar. Um alle, für einen Wecker benötigten, Variablen in einer Datenstruktur zu speichern, ist die Folgende in Listing A.1 (Zeile 23-33) definiert:

Tabelle 5.2.: Datenstruktur `struct wecker`

wecker
active: boolean
hour: uint8_t
minute: uint8_t
repeat: uint8_t
weekday: uint8_t
weekdays: boolean[7]
light: uint8_t
volume: uint8_t
track: uint8_t

Mit Hilfe der Datenstruktur `wecker` sind mehrere Wecker, mit umfangreichen Einstellungen, initialisierbar. Die Variable `boolean active` beinhaltet die Information, ob der Wecker ein- oder ausgeschaltet ist. In den Variablen `uint8_t hour` und `uint8_t minute` ist die eingestellte Weckzeit hinterlegt. Die Art der Wiederholung des Weckers, wie z.B. einmalig, täglich, wöchentlich, Montag bis Freitag oder Wochenende, ist in `uint8_t repeat` gespeichert. Diese Variable dient hauptsächlich für die Anzeige auf dem LCD. Die Variable `uint8_t weekday` beinhaltet den Wochentag, an dem der Wecker ansteht und findet nur bei einmaliger und wöchentlicher Wiederholung Anwendung. Das Array `boolean weekdays[7]` kam nachträglich zur Datenstruktur hinzu, um die Überprüfung, ob ein Alarm ansteht, zu vereinfachen. Dieses besteht aus sieben Elementen, wobei jedes Element für einen Wochentag steht. Wenn ein Element auf den Wert `true` gesetzt ist, ist der Wecker für diesen Wochentag konfiguriert. Der Lichtvorlauf bzw. wann der Weckvorgang mit Licht startet, ist in der Variable `uint8_t light` hinterlegt. Dieser Wert stellt dar, wie viele Minuten vor der Weckzeit der Lichtvorlauf startet. Die Variable `uint8_t volume` beinhaltet die Lautstärke des akustischen Signals. Jedoch hat diese Variable bei der Software des Lichtweckers noch keinen Anwendungsfall. Das individuelle akustische Signal ist in der Variable `uint8_t track` gespeichert. Diese Variable ist die Nummer der, auf der microSD gespeicherten, MP3 Datei, die als Weckton abzuspielen ist.

Die Datenstruktur ist in Listing A.1 (Zeile 79-83) genutzt, um die vier konfigurierbaren Wecker und den nächsten anstehenden Wecker zu initialisieren.

5.4.2. Akustische Weckfunktion

Die akustische Weckfunktion ist in den Funktionen `wakeUpTime(wecker wecker)`, in Listing A.2 (Zeile 2-6), und `wakeUp()`, in Listing A.2 (Zeile 29-37), umgesetzt. Die `wakeUpTime(wecker wecker)`-Funktion, wie auch als Ablaufdiagramm in Abbildung 5.7 dargestellt, prüft, ob die Weckzeit, der übergebene `wecker` Datenstruktur, fällig ist. Dabei überprüft die Funktion sowohl, ob der Wecker ein- oder ausgeschaltet ist, als auch, ob der Wecker für den aktuellen Wochentag konfiguriert ist. Mit Hilfe der Wochentagüberprüfung sind die Wochenendabschaltung, „Wochenendwecker“ und Wecker mit wöchentlicher Wiederholung, umgesetzt. Diese Überprüfung ist später hinzugefügt, da zunächst nur ein einfacher Wecker implementiert wurde. Bei Fälligkeit der Weckzeit, gibt die Funktion den Wert `true` zurück, sonst ist der Rückgabewert `false`.

Die Wiedergabe des, durch den Benutzer konfigurierten, akustischen Signals erfolgt in der Funktion `wakeUp()`. Dabei wird, wie auch im Ablaufdiagramm in Abbildung 5.8 zu erkennen, die maximale Lautstärke des MP3 Player Moduls eingestellt und der individuelle Wecktitel, von der microSD, wiedergegeben.

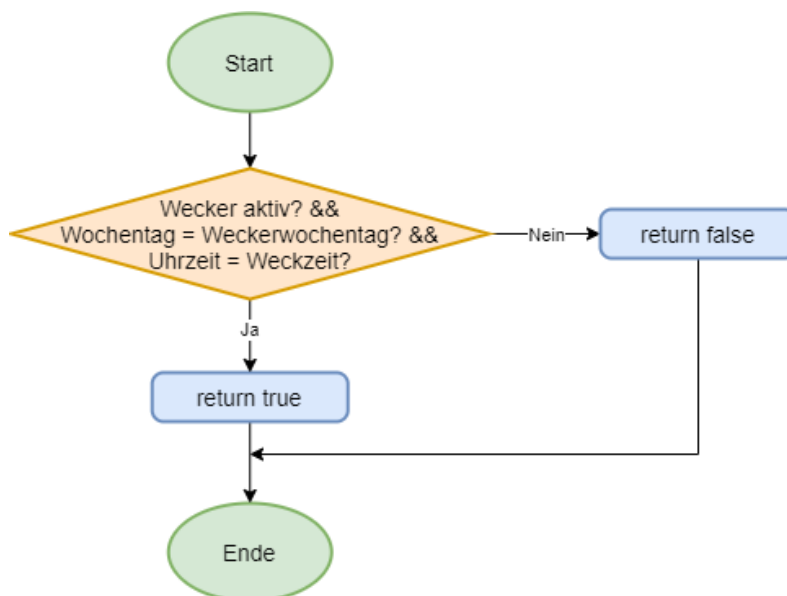


Abbildung 5.7.: Ablaufdiagramm der Funktion `wakeUpTime` (wecker wecker)

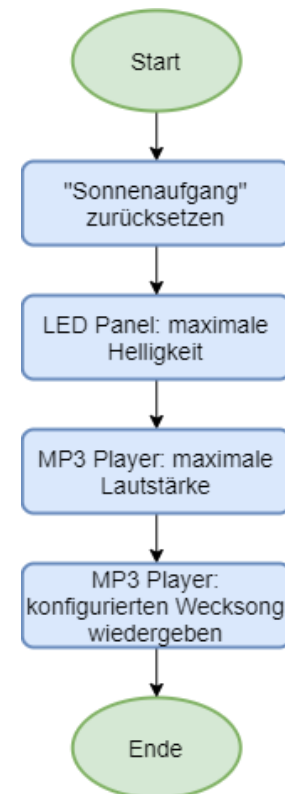


Abbildung 5.8.: Ablaufdiagramm der Funktion `wakeUp()`

5.4.3. Lichtweckfunktion

Hauptfunktion des Lichtweckers ist das Wecken mit Hilfe von Licht. Dabei ist, wie in Kapitel 2 geschildert, eine Lichtfunktion umzusetzen, die einen Sonnenaufgang nachahmt. Dieser beginnt eine definierte Anzahl von Minuten vor der gewünschten Weckzeit und erreicht die maximale Helligkeit zur Weckzeit. Die Vorlaufzeit, in Minuten, wird durch den Benutzer, über die Menüstruktur, definiert und in der Variable `uint8_t light`, der Datenstruktur `wecker`, gespeichert. Die Lichtweckfunktion ist in den Funktionen `wakeUpLight(wecker wecker)`, `sunrise()` und `wakeUp()` umgesetzt.

Die Funktion `wakeUpLight(wecker wecker)`, in Listing A.2 (Zeile 9-26), überprüft, ob der Lichtvorlauf, bei dem übergebenen Wecker fällig ist. Die Funktion berechnet, wie im Ablaufdiagramm in Abbildung 5.9 gezeigt, zunächst die Weckzeit in Minuten und zieht davon die individuelle Lichtvorlaufzeit, des Weckers, ab. Anschließend überprüft die Funktion, ob der berechnete Wert kleiner oder größer gleich Null ist. Diese Abfrage ist notwendig, da der Lichtvorlauf, z.B. bei einer konfigurierten Weckzeit um 0:20 mit einer Lichtvorlaufzeit von 40 Minuten, schon am Vortag, im Beispiel um 23:40, starten muss. Daraufhin erfolgt die Überprüfung, ähnlich wie bei der Funktion `wakeUpTime(wecker wecker)`, ob der Beginn des Lichtvorlaufs fällig ist. Bei Fälligkeit des Lichtvorlaufs, werden die Parameter des Weckers, als nächster anstehender Wecker in der Variable `wecker weckerX`, zwischengespeichert und die Funktion gibt den Wert `true` zurück. Wenn der Lichtvorlauf nicht fällig ist, so ist der Rückgabewert `false`.

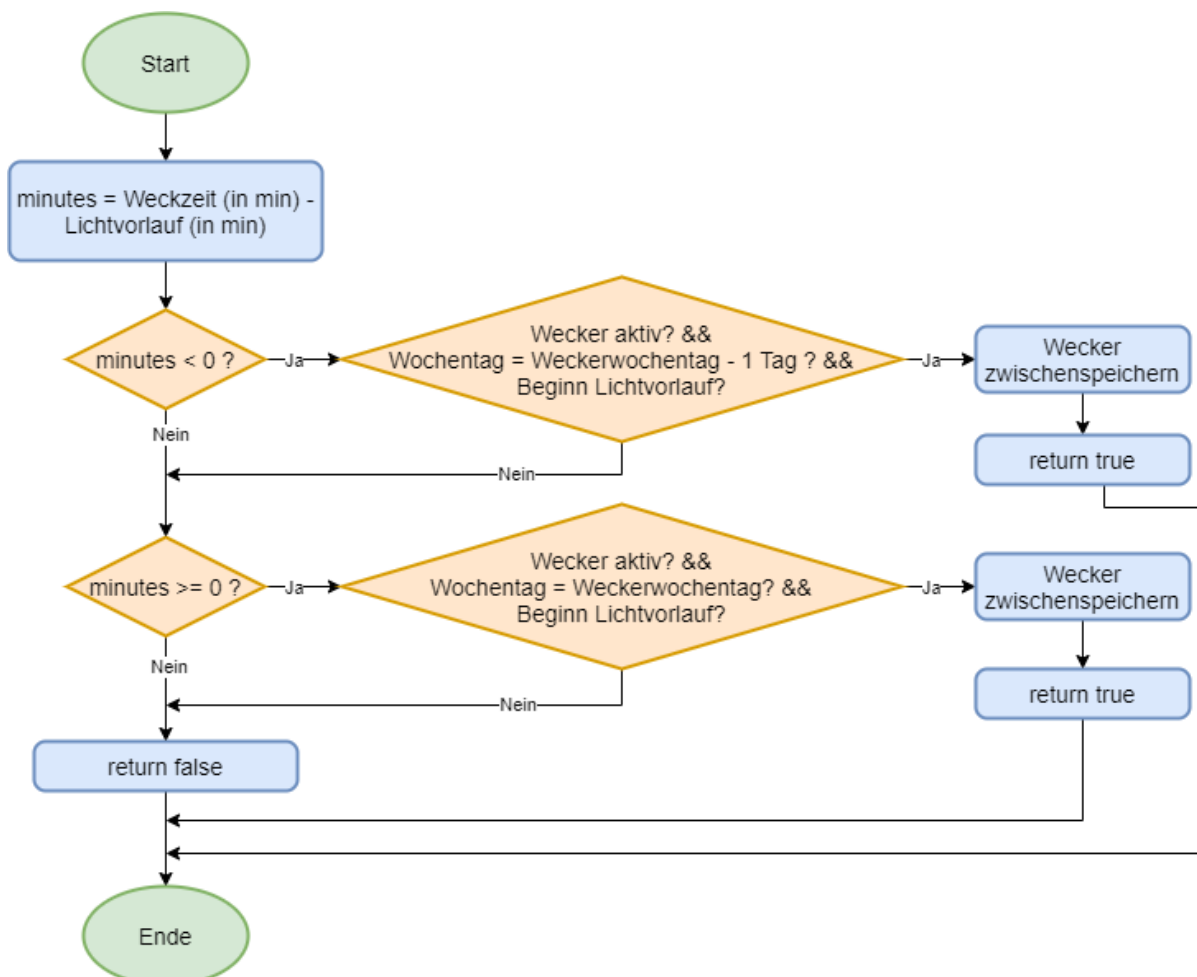


Abbildung 5.9.: Ablaufdiagramm der Funktion `wakeUpLight(wecker wecker)`

Der sonnenaufgangähnliche Lichtverlauf ist in der `sunrise()`-Funktion, in Listing A.2 (Zeile 40-76), umgesetzt. Innerhalb von 120 Zyklen realisiert die Funktion den Lichtverlauf auf dem LED Panel. Die Zeit in Sekunden, zwischen jedem Intervall, ist die einheitenlose Lichtvorlaufzeit, des nächsten anstehenden Weckers, geteilt durch zwei. Im Verlauf des „Sonnenaufgangs“ wird die Helligkeit des LED Panels kontinuierlich, von der minimalen bis zur nahezu maximalen Helligkeit, erhöht. Dabei durchläuft das LED Panel zeitgleich und kontinuierlich die Farben Rot, Orange, Gelb und Weiß, welche einem Sonnenaufgang ähneln.

Bei der Fälligkeit einer Weckzeit, wird, wie in Unterabschnitt 5.4.2 beschrieben, die Funktion `wakeUp()`, in Listing A.2 (Zeile 29-37), aufgerufen. Diese setzt, wie auch im Ablaufdiagramm in Abbildung 5.8 dargestellt, die Parameter der Funktion `sunrise()`, zurück und somit auch den „Sonnenaufgang“. Zudem wird die Farbe des LED Panels auf Weiß, mit maximaler Helligkeit, gestellt.

5.4.4. Beenden der Wecksignale

Zum Zurücksetzen des visuellen und akustischen Alarms dient die Funktion `alarmOff`, in Listing A.2 (Zeile 79-82). In dieser wird, wie im Ablaufdiagramm in Abbildung 5.10 dargestellt, zunächst das LED Panel auf die Einstellungen, vor Beginn des Lichtvorlaufs, zurückgesetzt. Des Weiteren wird die Wiedergabe des akustischen Signals, über den MP3 Player, pausiert.

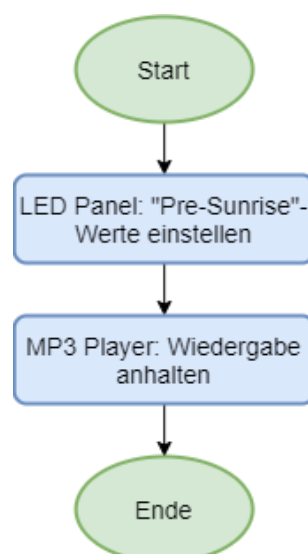


Abbildung 5.10.: Ablaufdiagramm der Funktion `alarmOff()`

5.5. Zusammenführung der Einzelkomponenten und Weckfunktionen

Dieser Abschnitt der Studienarbeit thematisiert die Zusammenführung der, zuvor in Abschnitt 5.3 und Abschnitt 5.4 implementierten, Einzelkomponenten und Weckfunktionen. Dabei erfolgt die Zusammenführung zum einen über eine Menüstruktur, sodass der Benutzer die verschiedenen Einzelkomponenten und Weckfunktionen konfigurieren und nutzen kann. Zum anderen findet eine Zusammenführung in der `loop()`-Funktion des Sketches statt, um alle Funktionen lauffähig und logisch miteinander zu verknüpfen. Abschließend wird eine Laufzeitmessung der `loop()`-Funktion durchgeführt, um die Einhaltung der Zeitbedingungen zu testen.

5.5.1. Menüstruktur

Die umgesetzte Menüstruktur dient zur Nutzung und Einstellung bzw. Konfiguration, der einzelnen Komponenten, durch den Benutzer. Somit wird ermöglicht, dass dieser über das LCD, Informationen und Einstellungen angezeigt bekommt und diese, mit Hilfe der Taster anpassen kann. Bestandteil der Menüstruktur sind dabei die beiden globalen Variablen `uint8_t page` und `uint8_t line`, in Listing A.1 (Zeile 45 f.), sowie die `switch`-Anweisung, in Listing A.1 (Zeile 180-230), die `selectEvent()`-Funktion, in Listing A.7, und die Funktion `backEvent()`, in Listing A.3.

Die aktuelle Menüebene ist dabei in der globalen Variable `uint8_t page` gespeichert. In der globalen Variable `uint8_t line` ist die, aktuell ausgewählte, Zeile hinterlegt. Somit ist im Sketch jederzeit bekannt, wo sich der Benutzer, in der Menüstruktur, befindet. Die Anzeige der aktuellen Uhrzeit und des aktuellen Datums ist die höchste Menüebene (page 0). Die Variable `uint8_t line`, der ausgewählte Menüpunkt, kann der Benutzer durch Drücken des UP-Tasters (`pressedUP()` in Listing A.1 (Zeile 102-109)) bzw. DOWN-Tasters (`pressedDOWN()` in Listing A.1 (Zeile 92-99)), ändern.

Der Taster SELECT hingegen dient dazu, eine Aktion durchzuführen. Die jeweilige auszuführende Aktion, welche spezifisch für jede Menüebene und jeden Menüpunkt ist, wird über die Funktion `selectEvent()` ausgewählt. Je nach aktuellem Wert der Variablen `uint8_t page` und `uint8_t line`, gelangt der Benutzer in eine tiefere Menüebene oder es wird eine Konfiguration ausgeführt.

Das Drücken des BACK-Tasters führt zum Aufruf der Funktion `backEvent()`. Diese Funktion ändert die Variable `uint8_t page` auf den Wert der nächsthöheren Menüebene und speichert Einstellungen, wenn dies in der jeweiligen Menüebene möglich ist, im EEPROM.

Die visuelle Darstellung, der aktuellen Menüebene und des ausgewählten Menüpunktes, ist mit der `switch`-Anweisung, in Listing A.1 (Zeile 180-230), realisiert. Über diese wird die jeweilige Funktion, zum Anzeigen von Daten auf dem LCD, aus Listing A.9, aufgerufen.

Die Menüstruktur, und deren Funktionsweise, ist grafisch abstrahiert in Abbildung B.1, in Anhang B, abgebildet. Dabei stellt jede Box, ohne Füllung, eine Menüebene (`uint8_t page`), dar. Die Menüebenen, der vier einstellbaren Wecker, sind hier als eine Box zusammengefasst (`page 6/7/8/9` und `page 10/11/12/23`), da sich lediglich die angezeigten Variablen ändern. Rechtecke mit blauer oder grüner Füllung stehen für eine auszuführende Aktion. Der Unterschied zwischen blauen und grünen Rechtecken besteht darin, dass bei grünen Rechtecken die Änderung bzw. Einstellung, durch die jeweilige Aktion, im EEPROM gespeichert wird. Bei blauen Rechtecken wird die Aktion lediglich ausgeführt, wobei eine Änderung bzw. Einstellung nicht, im EEPROM, gespeichert wird. Die Funktionen `selectEvent()` und `backEvent()` sind in der grafischen Darstellung durch Pfeile und Rechtecke dargestellt, da diese die nächsthöhere oder nächsttiefere Menüebene, sowie die auszuführenden Aktionen, bestimmen.

5.5.2. `loop()`-Funktion

Die `loop()`-Funktion ist, wie in Abschnitt 5.2 beschrieben, eine Endlosschleife und ist Basis eines jeden Sketches. Dabei ruft die Endlosschleife Funktionen, zyklisch oder ereignisbasiert, auf und bildet somit die logische Verknüpfung zwischen Menüstruktur, Einzelkomponenten und Weckfunktionen. Die Funktionsweise ist grafisch, als Ablaufdiagramm, in Abbildung C.1, in Anhang C, dargestellt.

Zu Beginn der `loop()`-Funktion wird geprüft, ob ein Taster gedrückt ist. Bei gedrücktem SELECT- bzw. BACK-Taster wird, wie in Unterabschnitt 5.5.1 erläutert, die Funktion `selectEvent()` bzw. `backEvent()` aufgerufen und die jeweilige Aktion, aus Listing A.7 bzw. Listing A.3, ausgeführt. Anschließend findet eine Aktualisierung, der LCD-Ausgabe, statt. Dies geschieht mit Hilfe einer `switch`-Anweisung, welche die Ausgabefunktion, der jeweiligen Menüebene, aufruft.

Wenn kein Taster, zur Navigation, gedrückt ist, der Benutzer sich auf der höchsten Menüebene (page 0) befindet, die Zeit gestellt ist und seit der letzten Anzeige von Uhrzeit, sowie Datum, eine Sekunde vergangen ist, wird die Funktion `digitalClockDisplay()` aufgerufen. Diese gibt die aktuelle Uhrzeit, inklusive der aktuellen Sekunde, und das aktuelle Datum mit Wochentag, über das LCD, aus.

Anschließend findet die Überprüfung, des ALARM-Tasters, statt. Wenn dieser gedrückt ist, wird die Funktion `alarmOff()`, die das akustische und visuelle Wecksignal beendet, aufgerufen.

Wenn das Lichtwecken freigegeben ist, wird die Funktion `sunrise()`, aufgerufen. Diese realisiert den visuellen Lichtweckvorgang, durch wiederholte Ausführung.

Daraufhin erfolgt die Fälligkeitsprüfung aller vier Wecker bzgl. des Lichtweckens und der Weckzeit. Jedoch erfolgt die Prüfung nur, wenn seit der letzten Überprüfung eine Minute vergangen ist. Bei Fälligkeit des Lichtweckens erfolgt eine Freigabe, sodass der „Sonnenaufgang“ im nächsten Durchlauf, der `loop()`, startet. Jedoch wird diese Freigabe nicht im EEPROM gespeichert, sodass die Freigabe, nach einem Neustart, zurückgesetzt ist. Zudem müssen aufeinanderfolgende Wecker eine zeitliche Differenz größer oder gleich des Lichtvorlaufs haben, um die Fälligkeit des Lichtweckens zu erkennen. Bei der Fälligkeit einer Weckzeit hingegen, findet ein Aufruf der Funktion `wakeUp()` statt. Diese startet das akustische Wecksignal und stellt das LED Panel auf volle Helligkeit. Diese Fälligkeit wird auch ausgelöst, wenn aufeinanderfolgende Weckzeiten einen zeitlichen Abstand von einer Minute haben.

Dieser Ablauf, der `loop()`-Funktion, wird dauerhaft und ohne Ende durchlaufen.

5.5.3. Test der zyklischen Ausführungszeit

Die zyklische Ausführungszeit des Sketches, insbesondere der `loop()`-Funktion, hat, bei dem Lichtwecker, große Relevanz. Wie in Abschnitt 4.8 diskutiert, handelt es sich beim Lichtwecker um feste Echtzeit, wobei die Anzeige der Uhrzeit und die Weckfunktion absoluten Zeitbedingungen unterliegen. Die Anzeige der Uhrzeit bedarf eine sekundengenaue, die Weckfunktion eine minutengenaue, Ausführung. Daraus folgt, dass die Ausführungszeit, der `loop()`-Funktion, kleiner einer Sekunde, zur Einhaltung der absoluten Zeitbedingungen, erforderlich ist.

Mit Hilfe des Quellcodes, in Listing 5.3, ist die Ausführungszeit, der `loop()`-Funktion, messbar. Zudem wird die gemessene Zeit über den seriellen Monitor ausgegeben. Dieser Quellcode kommt auch im Sketch des Lichtweckers zum Einsatz, jedoch ist dieser in der finalen Version, in Listing A.1 (Zeile 35, 265 f.), kommentiert. Vor dem Laden des finalen Sketches, wurde die Ausführungszeit von 1000000 Funktionsdurchläufen, ermittelt. Während der Messung wurden die Bedienelemente betätigt, sowie das LED Panel konfiguriert. Aufgrund der Menge der Messwert, sind diese in Anhang D, dem digitalen Anhang, in der Excel-Datei „Messwerte_Ausfuehrungszeit_loop.xlsx“, gespeichert.

Listing 5.3: Messung und Ausgabe der Ausführungszeit von `loop()`

```
1 unsigned long stopwatch;
2
3 void loop() {
4   //hier steht der Code, dessen Laufzeit gemessen wird
5
6   Serial.println(micros()-stopwatch);
7   stopwatch = micros();
8 }
```

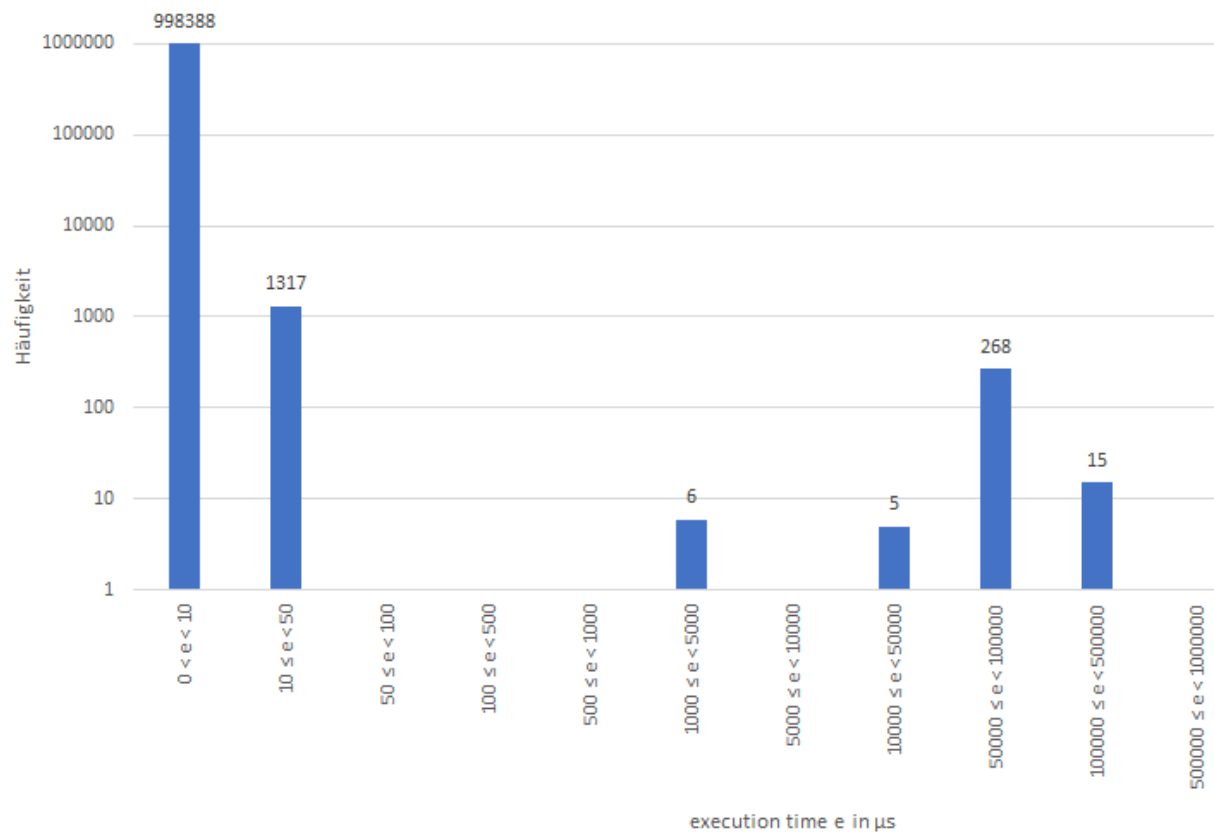
Nach der Ermittlung der Messwerte, erfolgte die Auswertung mit Hilfe von Microsoft Excel. Dabei wurden die Messwerte, wie in Tabelle 5.3, Klassen zugeordnet. Mit Hilfe der Klassen, lassen sich die Messwerte, wie in Abbildung 5.11 zu sehen, in einem Histogramm darstellen.

Tabelle 5.3.: Laufzeit der `loop()`-Funktion bei 1000000 Durchläufen

Execution time e in μs	Häufigkeit
$0 < e < 10$	998388
$10 \leq e < 50$	1317
$50 \leq e < 100$	0
$100 \leq e < 500$	0
$500 \leq e < 1000$	0
$1000 \leq e < 5000$	6
$5000 \leq e < 10000$	0
$10000 \leq e < 50000$	5
$50000 \leq e < 100000$	268
$100000 \leq e < 500000$	15
$500000 \leq e < 1000000$	0
$1000000 \leq e$	0

Sowohl anhand der klassifizierten Messwerte, in Tabelle 5.3, als auch anhand des Histogramms, in Abbildung 5.11, ist klar erkennbar, dass die Ausführungszeit, während der Versuchsmessungen, kleiner 500 Millisekunden ist. Des Weiteren ist im Histogramm eine typische Ausführungszeit von kleiner 50 μs , sowie größer 50 ms und kleiner gleich 100 ms, erkennbar. Die Ausführungszeit kleiner 50 μs ist als Durchlauf, ohne jegliche Verarbeitung oder Aktion, interpretierbar. Die Ausführungszeit größer 50 ms und kleiner gleich 100 ms hingegen, deutet auf die Ausführung einer Aktion bzw. Überprüfung einer Fälligkeit hin.

In der Messreihe sind die Zeitbedingungen des Lichtweckers nicht verletzt. Jedoch sind weitere Messreihen durchzuführen, welche, im Speziellen, die Ausführungszeiten der einzelnen Funktionen untersuchen. Somit ist eine Aussage mit höherer Signifikanz möglich.

Abbildung 5.11.: Histogramm zur Ausführungszeit der `loop()`-Funktion

6. Zusammenfassung

In diesem Kapitel ist das Fazit der Studienarbeit I, „Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion (Software)“, formuliert, wobei die Umsetzung der Software reflektiert ist. Zudem wird ein Ausblick gegeben, wie das Projekt zukünftig weitergeführt, erweitert und verbessert werden kann.

6.1. Fazit

Abschließend ist festzustellen, dass es im Zeitraum von 11 Wochen gelungen ist, die „Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion (Software)“, durchzuführen. Dabei wurden die notwendigen Komponenten und Softwarebibliotheken, in der theoretischen Vorbetrachtung (Kapitel 4), evaluiert und ausgewählt. Des Weiteren wurden die softwareseitigen Anforderungen, aus Kapitel 2, vollständig umgesetzt (Kapitel 5).

Für die Bedienung des Lichtweckers sind fünf Taster eingesetzt. Die intuitive Bedienung ist hier nicht optimal umgesetzt, da zum Einstellen, z.B. der Weckzeit, das wiederholte Drücken des SELECT-Tasters, um den jeweiligen Wert zu ändern, erforderlich ist. Für eine intuitivere Bedienung sind die UP- und DOWN-Taster, oder ein Drehwinkelgeber, für die Parametrierung, zu implementieren. Die Ausgabe von Informationen ist über ein LCD realisiert. Hier wird dem Benutzer die aktuelle Uhrzeit und das aktuelle Datum angezeigt. Die Synchronisation ist mit dem NTP, über eine Netzwerkverbindung, umgesetzt. Diese kann über ein WLAN-fähiges Drittgerät eingerichtet werden. Zudem ist eine Menüstruktur implementiert, über die es dem Nutzer möglich, vier Wecker, das LED Panel, die Zeit und den MP3 Player, zu konfigurieren bzw. anzusteuern.

Die vier Wecker verfügen über eine Wochenendabschaltung, sowie der Option den Wecker täglich, wöchentlich oder nur am Wochenende zu wiederholen. Zudem kann über die Menüstruktur ein beliebiger, auf der microSD gespeicherter Song, als Wecksound ausgewählt werden. Des Weiteren ist eine Lichtweckfunktion implementiert, die einen sonnenaufgangähnlichen Farb- und Helligkeitsverlauf über das LED Panel ausgibt. Der Beginn des Lichtweckens ist für jeden Wecker individuell einstellbar. Jegliche Einstellungen des Weckers sind im EEPROM gespeichert, sodass diese auch bei einem Neustart erhalten bleiben. Somit ist das Wecken durch ein akustisches Signal, auch nach einem Neustart, gewährleistet. Die Freigabe und der Fortschritt der Lichtweckfunktion hingegen, wird nicht im EEPROM gespeichert, wodurch die Lichtweckfunktion, bei einem Neustart, nicht fortgesetzt wird. Hier ist eine Anpassung der Lichtweckfunktion bzw. das Speichern der Freigabe und des Fortschritts notwendig.

Weiterhin kann der Benutzer das LED Panel, über die Menüstruktur, individuell, in Farbe und Helligkeit, einstellen und somit, als Beleuchtung, nutzen. Über die Menüstruktur ist auch die Bedienung des MP3 Players möglich, sodass dieser eigenständig, ohne fällige Weckzeit, einsetzbar ist. Des Weiteren ist, mit Hilfe der Menüstruktur, die händische Einstellung von Uhrzeit, Datum und Zeitzone, sowie die Synchronisation über das NTP, möglich. Da die Displayhintergrundbeleuchtung, bei geringer Umgebungsbeleuchtung, stört, ist diese ein- und ausschaltbar.

6.2. Ausblick

Der Lichtwecker verfügt über eine Vielzahl von Funktionalitäten, jedoch gibt es auch Verbesserungspotential. Um die Weiterentwicklung zu fördern, ist der Quellcode des Lichtweckers, in einem öffentlichen GitHub-Repository, unter <https://github.com/schackniss/esp32-lichtwecker>, verfügbar.

Als nächster Schritt sollte die Lichtweckfunktion angepasst werden, sodass der „Sonnenaufgang“, auch nach einem Neustart, weiterläuft. In Folge dessen ist eine Umstrukturierung der Wecker sinnvoll, sodass eine Klasse, anstatt einer Datenstruktur, implementiert wird. Dies würde die softwareseitige Handhabung erleichtern und die Anzahl der Wecker würde sich einfacher skalieren lassen.

Des Weiteren ist eine Optimierung der Bedienung sinnvoll, da diese, in der aktuellen Version, das Benutzererlebnis einschränkt. Hier würde sich eine verbesserte Logik der Taster oder die Verwendung eines Drehwinkelgebers anbieten. Da der Lichtwecker mit einem WLAN verbunden ist, ist auch eine Weboberfläche, zur Bedienung, denkbar.

In Bezug auf das LED Panel können weitere Einstellmöglichkeiten implementiert werden. Wenn zukünftig ein Webinterface umgesetzt wird, ist auch ein Colorpicker, zur Farb- und Helligkeitsauswahl, vorstellbar. Zudem könnte für das LED Panel eine Zeitschaltung, um ein Nachtlicht automatisch ein- bzw. auszuschalten, implementiert werden. Eine solche Zeitschaltung ist auch für die Hintergrundbeleuchtung des Displays wünschenswert. Somit würde sich diese, während der Schlafenszeit, ausschalten.

Da die Softwareentwicklung des Lichtweckers mit viel Spaß verbunden war, werde ich diese, in meiner Freizeit, fortführen.

Literaturverzeichnis

- [1] „Schlaf“, in *Brockhaus Enzyklopädie Online*, NE GmbH | Brockhaus, 2021. Adresse: <https://brockhaus.de/ecs/permalink/C0C62DF13A0630F91074CFC67945CB43.pdf> (besucht am 06.03.2021).
- [2] C. Blume, C. Garbazza und M. Spitschan, „Effects of light on human circadian rhythms, sleep and mood“, *Somnologie*, Jg. 23, Nr. 3, S. 147–156, 2019, ISSN: 1432-9123. DOI: 10.1007/s11818-019-00215-x.
- [3] Philips. „Sleep & Wake-up Light: Lichtwecker | Philips“. (2021), Adresse: <https://www.philips.de/c-e/pe/lichttherapie/wake-up-lights.html> (besucht am 06.03.2021).
- [4] R. Gilg, *Entwicklung eines Lichtweckers mit Uhren- und Alarmfunktion - Hardware*, Mannheim, 2021.
- [5] P. Scholz, *Softwareentwicklung eingebetteter Systeme: Grundlagen, Modellierung, Qualitätssicherung*, Ser. Xpert.press. Berlin: Springer, 2005, ISBN: 3540234055.
- [6] M. Wieden, *Chronobiologie im Personalmanagement: Wissen, wie Mitarbeiter ticken*, 2., aktualisierte und überarbeitete Auflage. Wiesbaden: Springer Gabler, 2016, ISBN: 9783658093556. DOI: 10.1007/978-3-658-09355-6.
- [7] H. Bernstein, *Mikrocontroller: Grundlagen der Hard- und Software der Mikrocontroller ATtiny2313, ATtiny26 und ATmega32*. Wiesbaden: Springer Vieweg, 2015, ISBN: 9783658028138. DOI: 10.1007/978-3-658-02813-8.
- [8] B. Spitzer, *Mikrorechnertechnik 1: Funktion und Aufbau eines Digitalrechners*, Mannheim, 2019.
- [9] Arduino, Hrsg. „Arduino Products“. (2021), Adresse: <https://www.arduino.cc/en/Products/Compare> (besucht am 09.03.2021).
- [10] Arduino, Hrsg. „a000062_featured_1_1.jpg (520×330)“. (2020), Adresse: https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/520x330/604a3538c15e081937dbfbd20aa60aad/a/0/a000062_featured_1_1.jpg (besucht am 09.03.2021).

-
- [11] Arduino, Hrsg. „a000067_featured_4.jpg (520×330)“. (2020), Adresse: https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/520x330/604a3538c15e081937dbfbd20aa60aad/a/0/a000067_featured_4.jpg (besucht am 09.03.2021).
- [12] Espressif Systems, Hrsg. „ESP32 Series: Datasheet“. (2021), Adresse: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (besucht am 08.03.2021).
- [13] B. Spitzer, *Mikrorechner-technik 2: Anwendung und Programmierung von Embedded Systems*, Mannheim, 2020.
- [14] AZ-Delivery, Hrsg. „devkitmain1er_500x.jpg (500×500)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/devkitmain1er_500x.jpg?v=1615204254 (besucht am 09.03.2021).
- [15] „Zeit“, in *Brockhaus Enzyklopädie Online*, NE GmbH | Brockhaus, 2021. Adresse: <https://brockhaus.de/ecs/permalink/DF6FA274D36357215EC0422948AA792F.pdf>.
- [16] C. Riewerts. „Einführung in Realzeitsysteme“. (2015), Adresse: <http://wwwlehre.dhbw-stuttgart.de/~rie/RZS/Vorlesung/RealzeitVorlesungKap1.pdf> (besucht am 01.04.2021).
- [17] Elektronik-Kompendium.de, Hrsg. „NTP - Network Time Protocol“. (), Adresse: <http://wwwlehre.dhbw-stuttgart.de/~rie/RZS/Vorlesung/RealzeitVorlesungKap1.pdf> (besucht am 01.04.2021).
- [18] Arduino, Hrsg. „NTPClient“. (2019), Adresse: <https://github.com/arduino-libraries/NTPClient> (besucht am 30.03.2021).
- [19] P. Stoffregen. „Time“. (2019), Adresse: <https://github.com/PaulStoffregen/Time> (besucht am 30.03.2021).
- [20] AZ-Delivery, Hrsg. „0_96_Zoll_I2C_OLED_Display_front_500x.jpg (500×500)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/0_96_Zoll_I2C_OLED_Display_front_500x.jpg?v=1614790241 (besucht am 14.03.2021).
- [21] Adafruit Industries, Hrsg. „Adafruit_SSD1306“. (2021), Adresse: https://github.com/adafruit/Adafruit_SSD1306 (besucht am 14.03.2021).
- [22] J. Rickman. „LiquidCrystal_I2C“. (2021), Adresse: https://github.com/johnrickman/LiquidCrystal_I2C (besucht am 14.03.2021).

-
- [23] AZ-Delivery, Hrsg. „2.Frontwologo_500x.jpg (500×500)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/2.Frontwologo_500x.jpg?v=1597657394 (besucht am 14.03.2021).
- [24] AZ-Delivery, Hrsg. „KY-016 LED RGB Modul Datenblatt“. (2020), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/RGB_LED_Ring_37mm_Datenblatt_AZ-Delivery_Vertriebs_GmbH.pdf?v=1608471975 (besucht am 16.03.2021).
- [25] AZ-Delivery, Hrsg. „1.Main_1x_KY-016_500x.jpg (500×500)“. (), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/1.Main_1x_KY-016_500x.jpg?v=1607506078 (besucht am 16.03.2021).
- [26] Adafruit Industries, Hrsg. „Adafruit_NeoPixel“. (2020), Adresse: https://github.com/adafruit/Adafruit_NeoPixel (besucht am 14.03.2021).
- [27] AZ-Delivery, Hrsg. „RGB LED Ring WS2812B 12-Bit 37mm Datenblatt“. (), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/U64_LED_Matrix_Datenblatt_AZ-Delivery_Vertriebs_GmbH.pdf?v=1608116531 (besucht am 16.03.2021).
- [28] AZ-Delivery, Hrsg. „U64 LED Matrix Datenblatt“. (), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/U64_LED_Matrix_Datenblatt_AZ-Delivery_Vertriebs_GmbH.pdf?v=1608116531 (besucht am 16.03.2021).
- [29] AZ-Delivery, Hrsg. „Front_1ca3c96e-9282-4095-b3d5-59dbd47587be_500x.jpg (500×500)“. (), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/Front_1ca3c96e-9282-4095-b3d5-59dbd47587be_500x.jpg?v=1602684314 (besucht am 16.03.2021).
- [30] AZ-Delivery, Hrsg. „U-64-LED-Panel_front_500x.jpg (500×500)“. (), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/KY-016_LED_RGB_Modul_AZ-Delivery_Vertriebs_GmbH.pdf?6046459410546974998 (besucht am 16.03.2021).
- [31] AZ-Delivery, Hrsg. „ky-004-taster-modul-sensor-taste-kopf-schalter-schlussschalter-fur-arduinosenzoraz-deliveryaz-delivery-23972661_900x.jpg (900×900)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/Button_Modul_Datenblatt.pdf?532653108151546991 (besucht am 29.03.2021).
- [32] AZ-Delivery, Hrsg. „Drehimpulsgeber_Modul_Datenblatt“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/Drehimpulsgeber_Modul_Datenblatt.pdf?349756184529908641 (besucht am 29.03.2021).
-

-
- [33] AZ-Delivery, Hrsg. „Button Modul Datenblatt“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/Button_Modul_Datenblatt.pdf?532653108151546991 (besucht am 29.03.2021).
- [34] AZ-Delivery, Hrsg. „1.Main_1x_KY-040DrehimpulsgeberModul_500x.jpg (500×500)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/1.Main_1x_KY-040DrehimpulsgeberModul_500x.jpg?v=1607514482 (besucht am 29.03.2021).
- [35] P. Schnabel, Hrsg. „Prellfreier Schalter / Taster entprellen“. (2021), Adresse: <https://www.elektronik-kompodium.de/sites/dig/0210223.htm> (besucht am 29.03.2021).
- [36] Mikrocontroller.net, Hrsg. „Entprellung“. (2021), Adresse: <https://www.mikrocontroller.net/articles/Entprellung> (besucht am 29.03.2021).
- [37] Mikrocontroller.net, Hrsg. „Entprellen.png“. (2004), Adresse: <https://www.mikrocontroller.net/articles/Datei:Entprellen.png> (besucht am 29.03.2021).
- [38] AZ-Delivery, Hrsg. „Buzzer Modul aktiv Datenblatt“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/Buzzer_Modul_passiv_Datenblatt.pdf?10257969702650787294 (besucht am 29.03.2021).
- [39] AZ-Delivery, Hrsg. „Buzzer Modul passiv Datenblatt“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/Buzzer_Modul_passiv_Datenblatt.pdf?10257969702650787294 (besucht am 29.03.2021).
- [40] AZ-Delivery, Hrsg. „KY-012_Buzzer_Modul_aktiv_500x.jpg (500×500)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/KY-012_Buzzer_Modul_Aktiv_Datenblatt_AZ-Delivery_Vertriebs_GmbH.pdf?v=1605115550 (besucht am 29.03.2021).
- [41] AZ-Delivery, Hrsg. „KY-006_Buzzer_Modul_passiv_500x.jpg (500×500)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/KY-006_Buzzer_Modul_passiv_500x.jpg?v=1581330116 (besucht am 29.03.2021).
- [42] AZ-Delivery, Hrsg. „MP3 Player Modul Datenblatt“. (), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/MP3_Player_Modul_Datenblatt.pdf?10537896017176417241 (besucht am 29.03.2021).
- [43] DFRobot, Hrsg. „DFRobotDFPlayerMini: DFPlayer - A Mini MP3 Player For Arduino“. (2016), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/files/MP3_Player_Modul_Datenblatt.pdf?10537896017176417241 (besucht am 29.03.2021).
-

-
- [44] A. Vu, „Musikdateien mit dem MP3 Player Modul abspielen“, 2018. Adresse: <https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/erste-schritte-mit-dem-mp3-player-modul> (besucht am 29.03.2021).
- [45] AZ-Delivery, Hrsg. „1.Main_1x_mp3playermodul_500x.jpg (500×500)“. (2021), Adresse: https://cdn.shopify.com/s/files/1/1509/1638/products/1.Main_1x_mp3playermodul_500x.jpg?v=1606144996 (besucht am 29.03.2021).
- [46] H. Wörn und U. Brinkschulte, *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen ; mit 32 Tabellen*, Ser. eXamen.press. Berlin: Springer, 2005, ISBN: 6610621292.
- [47] P. Metzger, *Entwicklung eines Licht-Weckers mit Uhren- und Alarmfunktion: Software*, Mannheim, 2019.
- [48] R. Crutzen. „Arduino Wakeup Lights“. (2016), Adresse: <https://www.instructables.com/Wakeup-Lights/> (besucht am 30.03.2021).
- [49] T. K. Hareendran. „DIY Sunrise Alarm Using Arduino“. (2019), Adresse: <https://www.electroschematics.com/diy-sunrise-alarm-using-arduino/> (besucht am 30.03.2021).
- [50] S. Münzel. „LichtWecker“. (2020), Adresse: <https://github.com/s-muenzel/LichtWecker> (besucht am 30.03.2021).
- [51] A. Vu. „ESP32 jetzt über den Boardverwalter installieren“. AZ-Delivery, Hrsg. (2018), Adresse: <https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/esp32-jetzt-mit-boardverwalter-installieren> (besucht am 31.03.2021).
- [52] Arduino, Hrsg. „Sketch“. (2021), Adresse: <https://www.arduino.cc/en/tutorial/sketch> (besucht am 31.03.2021).
- [53] Arduino, Hrsg. „#include“. (2021), Adresse: <https://www.arduino.cc/reference/de/language/structure/further-syntax/include/> (besucht am 01.04.2021).
- [54] Arduino, Hrsg. „#define“. (2021), Adresse: <https://www.arduino.cc/reference/de/language/structure/further-syntax/define/> (besucht am 01.04.2021).
- [55] Arduino, Hrsg. „setup()“. (2021), Adresse: <https://www.arduino.cc/reference/de/language/structure/sketch/setup/> (besucht am 01.04.2021).
- [56] Arduino, Hrsg. „loop()“. (2021), Adresse: <https://www.arduino.cc/reference/de/language/structure/sketch/loop/> (besucht am 01.04.2021).
-

- [57] tzapu. „WiFiManager“. (2020), Adresse: <https://github.com/tzapu/WiFiManager> (besucht am 01.04.2021).

A. Anhang: Quellcode des Lichtweckers

Dieser Teil des Anhangs umfasst den Arduino Quellcode des Lichtweckers. Dieser ist auch, unter <https://github.com/schackniss/esp32-lichtwecker>, verfügbar.

A.1. 2021-03-27_V28.ino

Listing A.1: 2021-03-27_V28.ino

```
1 #include <Wire.h> //Lib for I2C communication
2 #include <LiquidCrystal_I2C.h> //Lib for LCD via I2C https://github.com
   /johnrickman/LiquidCrystal_I2C
3 #include <Adafruit_NeoPixel.h> //Lib for LED Neopixel Matrix Panel
   https://github.com/adafruit/Adafruit_NeoPixel
4 #include <WiFiUdp.h> //Lib for UDP connection
5 #include <WiFiManager.h> //Lib to setup wifi connection via Smartphone/
   PC https://github.com/tzapu/WiFiManager
6 #include <TimeLib.h> //Time Lib including NTP sync https://github.com/
   PaulStoffregen/Time
7 #include <DFRobotDFPlayerMini.h> //Lib for MP3 Player https://github.
   com/DFRobot/DFRobotDFPlayerMini
8 #include <EEPROM.h> //Flash memory lib https://github.com/espressif/
   arduino-esp32/tree/master/libraries/EEPROM
9
10 #define DOWN 26 //define input pin - down button
11 #define SELECT 23 //define input pin - select button
12 #define BACK 27 //define input pin - back button
13 #define UP 25 //define input pin - up button
14 #define ALARM 18 //define input pin - alarm button
15 #define LED 19 //define pin to communicate with LED MAtrix Panel
16 #define SCL 22 //define I2C pin - clock
```

```
17 #define SDA 21 //define I2C pin - data
18 #define NUMPIXELS 64 //define number of LED pixels
19 #define EEPROM_SIZE 64 //define number of bytes to access from EEPROM
20 #define BOUNCE 200 //define debounce time
21
22 //datastructure to set different alarm clocks
23 struct wecker {
24     boolean active;
25     uint8_t hour; //0...23
26     uint8_t minute; //0...59
27     uint8_t repeat; //0:einmalig 1:taeglich 2:woechentlich 3:Mo-Fr 4:
        Wochenende
28     uint8_t weekday; //0:- 1:So 2:Mo ... 7:Sa
29     boolean weekdays[7]; //0:So 1:Mo ... 6:Sa
30     uint8_t light; //Lichtvorlaufzeit: 20, 30, 40 min
31     uint8_t volume;
32     uint8_t track; //1...Anzahl Tracks
33 };
34
35 //unsigned long stopwatch; //variable for execution time test
36
37 LiquidCrystal_I2C display(0x27,20,4); //LCD - set the address to 0x27
        for a 20 chars and 4 line display
38 Adafruit_NeoPixel pixels(NUMPIXELS, LED, NEO_GRB + NEO_KHZ800); //LED
        Neopixel Matrix Panel
39 WiFiUDP Udp; //UDP connection
40 unsigned int localPort = 8888; // local port to listen for UDP packets
41
42 DFRobotDFPlayerMini mp3Player; //MP3 Player
43 uint8_t volume = 10; //MP3 Player Volume 0...30
44
45 uint8_t page = 0; //currently display page 0...16
46 uint8_t line = 0; //current cursor position on page 0...3
47
48 uint8_t prevMinute = 0; //value of the previous minute 0...59 -> needed
        for wakeUpLigt(wecker wecker) and wakeUpTime(wecker wecker)
49
50 //Light color
51 uint8_t red = 0; //red value of LED Matrix
52 uint8_t green = 0; //green value of LED Matrix
53 uint8_t blue = 0; //blue value of LED Matrix
54
55 time_t prevDisplay = 0; //when the digital clock was displayed
56
```

```
57 const char* wocheTag[]={ "-", "So", "Mo", "Di", "Mi", "Do", "Fr", "Sa"};
    //array of weekday names
58 const char* wiederholung[]={ "einmalig", "taeglich", "woechentlich", "Mo-
    Fr", "Wochenende"}; //array of alarm clock repeat values
59
60 const char ntpServerName[] = "europe.pool.ntp.org"; //NTP server address
61 int timeZone = 1; //timezone: Central European Time
62
63 const int NTP_PACKET_SIZE = 48; // NTP time is in the first 48 bytes of
    message
64 byte packetBuffer[NTP_PACKET_SIZE]; //buffer to hold incoming & outgoing
    packets
65
66 boolean pressed = false; //value if a button is pressed
67 boolean prevPressed = false; //value if a button was previously pressed
68 boolean pressedSelect = false; //value if Select button is pressed
69 boolean pressedBack = false; //value if Back button is pressed
70 boolean pressedAlarm = false; //value if Alarm button is pressed
71 boolean licht = false; //value if LED Matrix is on or off
72 boolean displayLicht = true; //value if LCD Backlight is on or off
73
74 boolean fade = false; //value if wakeup light is on or off
75 uint8_t counter = 0; //counter for wakeup light 0...120
76
77 unsigned long oldTime = millis(); //value in ms since last change of
    Button input value to debounce buttons
78
79 wecker wecker1; //alarm clock 1
80 wecker wecker2; //alarm clock 2
81 wecker wecker3; //alarm clock 3
82 wecker wecker4; //alarm clock 4
83 wecker weckerX; //values of the due alarm clock
84
85 uint8_t hrSetup = 0; //manual time setup - hour
86 uint8_t minSetup = 0; //manual time setup - minute
87 uint8_t daySetup = 1; //manual date setup - day
88 uint8_t monthSetup = 1; //manual date setup - month
89 int yrSetup = 2021; //manual date setup - year
90
91 //ISR: Down-Button
92 void IRAM_ATTR pressedDOWN() {
93     //following if-statement is debouncing the button
94     if(millis()-oldTime > BOUNCE) {
95         if (page > 0 && line < 3) line++;
96         pressed = !pressed;
```

```
97     oldTime = millis();
98   }
99 }
100
101 //ISR: Up-Button
102 void IRAM_ATTR pressedUP() {
103   //following if-statement is debouncing the button
104   if(millis()-oldTime > BOUNCE) {
105     if (page > 0 && line > 0) line--;
106     pressed = !pressed;
107     oldTime = millis();
108   }
109 }
110
111 //ISR: Select-Button
112 void IRAM_ATTR pressedSELECT() {
113   //following if-statement is debouncing the button
114   if(millis()-oldTime > BOUNCE) {
115     pressedSelect = true;
116     pressed = !pressed;
117     oldTime = millis();
118   }
119 }
120
121 //ISR: Back-Button
122 void IRAM_ATTR pressedBACK() {
123   //following if-statement is debouncing the button
124   if(millis()-oldTime > BOUNCE) {
125     pressedBack = true;
126     pressed = !pressed;
127     oldTime = millis();
128   }
129 }
130
131 //ISR: Alarm-Button
132 void IRAM_ATTR pressedALARM() {
133   //following if-statement is debouncing the button
134   if(millis()-oldTime > BOUNCE) {
135     pressedAlarm = true;
136     pressed = !pressed;
137     oldTime = millis();
138   }
139 }
140
141 //setup code to run once
```



```
142 void setup() {
143   Serial.begin(115200); //start serial connection to Serial Monitor
144   Serial2.begin(9600); //start serial connection to MP3 Player
145
146   input(); //configure pins as input and enable the internal pull-up
           resistors
147   output(); //configure pins as output
148   interrupt(); //configure interrupts
149   flash(); // initialize and configure EEPROM
150   lcd(); //configure LCD
151   light(); //read saved LED matrix values from flash and set light
152   wlan(); //connect to WiFi
153   udp(); //start UDP connection
154   ntp(); //set NTP timesync settings
155   mp3(); //start MP3 Player
156   initWecker(); //read saved alarm clock values from flash
157
158   display.clear();
159 }
160
161 //main code to run repeatedly
162 void loop() {
163   if (pressed != prevPressed) {
164     prevPressed = pressed;
165     display.clear();
166
167     //Select-Button is pressed
168     if (pressedSelect) {
169       selectEvent(); //Event, if Select-Button is pressed
170       pressedSelect = false;
171     }
172
173     //Back-Button is pressed
174     if (pressedBack) {
175       backEvent(); //Event, if Back-Button is pressed
176       pressedBack = false;
177     }
178
179     //display menu on LCD
180     switch (page) {
181       case 1:
182         printMenu();
183         break;
184       case 2:
185         printWecker();
```

```
186     break;
187     case 3:
188         printLicht();
189         break;
190     case 4:
191         printMusik();
192         break;
193     case 5:
194         printSonstiges();
195         break;
196     case 6:
197         printWeckerMenu(wecker1);
198         break;
199     case 7:
200         printWeckerMenu(wecker2);
201         break;
202     case 8:
203         printWeckerMenu(wecker3);
204         break;
205     case 9:
206         printWeckerMenu(wecker4);
207         break;
208     case 10:
209         printWeckerTime(wecker1);
210         break;
211     case 11:
212         printWeckerTime(wecker2);
213         break;
214     case 12:
215         printWeckerTime(wecker3);
216         break;
217     case 13:
218         printWeckerTime(wecker4);
219         break;
220     case 14:
221         printTimeDateSetup();
222         break;
223     case 15:
224         printTimeSetup();
225         break;
226     case 16:
227         printDateSetup();
228         break;
229     default: break;
230 }
```

```
231 }
232
233 //display current time and date on LCD, if the time is set, displayed
    page is 0 and 1 sec passed since last time-update
234 if (timeStatus() != timeNotSet && page == 0) {
235     if (now() != prevDisplay) { //update the display only if time has
        changed
236         prevDisplay = now();
237         digitalClockDisplay();
238     }
239 }
240
241 //Alarm-Button is pressed
242 if (pressedAlarm) {
243     alarmOff(); //turn off alarm (light and music)
244     pressedAlarm = false;
245 }
246
247 //wake-up light
248 if (fade) {
249     sunrise(); //wake-up light as sunrise
250 }
251
252 //check, if a alarm or wake-up light is due
253 if (prevMinute != minute()) {
254     prevMinute = minute();
255     if (wakeUpLight(wecker2) || wakeUpLight(wecker1) || wakeUpLight(
        wecker3) || wakeUpLight(wecker4)) {
256         fade = true;
257     }
258
259     if (wakeUpTime(wecker1) || wakeUpTime(wecker2) || wakeUpTime(wecker3
        ) || wakeUpTime(wecker4)) {
260         wakeUp();
261     }
262 }
263
264 //execution time test
265 //Serial.println(micros()-stopwatch);
266 //stopwatch = micros();
267 }
```

A.2. alarm.ino

Listing A.2: alarm.ino

```
1 //check, if alarm is due
2 boolean wakeUpTime(wecker wecker) {
3   if (wecker.active && wecker.weekdays[weekday()-1]
4       && wecker.hour == hour() && wecker.minute == minute()) return
5       true;
6   return false;
7 }
8 //check, if wake-up light needs to start; writes values of next due
9   alarm in weckerX
10 boolean wakeUpLight(wecker wecker) {
11   int minutes = (wecker.hour * 60) + wecker.minute - wecker.light;
12   if (minutes < 0) {
13     if (wecker.active && wecker.weekdays[weekday()-2]
14         && 23 == hour() && (60 + minutes) == minute()) {
15       weckerX = wecker;
16       return true;
17     }
18   }
19   if (minutes >= 0) {
20     if (wecker.active && wecker.weekdays[weekday()-1]
21         && (minutes / 60) == hour() && (minutes % 60) == minute()) {
22       weckerX = wecker;
23       return true;
24     }
25   }
26   return false;
27 }
28 //wake-up alarm (light, music)
29 void wakeUp() {
30   fade = false;
31   counter = 0;
32
33   setLight(255, 255, 255);
34
35   mp3Player.volume(30);
36   mp3Player.play(weckerX.track);
37 }
38
39 //wake-up light
```

```
40 void sunrise() {
41     static time_t prevSunrise = 0;
42     static uint8_t r = 0;
43     static uint8_t g = 0;
44     static uint8_t b = 0;
45
46     if (((now() - prevSunrise) >= (weckerX.light / 2))) {
47         prevSunrise = now();
48         Serial.println(now());
49
50         if (counter < 120) {
51             r += 2;
52             if (counter < 5)    g = r * 0.1;
53             if (counter >= 5  && counter < 10)  g = r * 0.2;
54             if (counter >= 10 && counter < 15)  g = r * 0.3;
55             if (counter >= 15 && counter < 20)  g = r * 0.4;
56             if (counter >= 20 && counter < 25)  g = r * 0.5;
57             if (counter >= 25 && counter < 30)  g = r * 0.6;
58             if (counter >= 30 && counter < 35)  g = r * 0.7;
59             if (counter >= 35 && counter < 40)  g = r * 0.8;
60             if (counter >= 40 && counter < 45)  g = r * 0.9;
61             if (counter >= 45) g = r;
62             if (counter >= 50 && counter < 100) b += 4;
63             if (counter >= 100) b = r;
64         }
65
66         setLight(r, g, b);
67         counter++;
68         if (counter >= 120) {
69             counter = 0;
70             r = 0;
71             g = 0;
72             b = 0;
73             fade = false;
74         }
75     }
76 }
77
78 //turn alarm off; stops music and sets light to original values
79 void alarmOff() {
80     setLight();
81     mp3Player.pause();
82 }
```

A.3. backEvent.ino

Listing A.3: backEvent.ino

```
1 //event/action, if Back-Button is pressed
2 void backEvent() {
3   switch (page) {
4     case 0:      //time and date
5       break;
6     case 1:      //menu
7       page = 0; //->time and date
8       break;
9     case 2:      //alarm menu
10      page = 1;  //->menu
11      line = 0;
12      break;
13     case 3:      //light
14      EEPROM.write(0, licht); //save licht (on/off) to EEPROM
15      EEPROM.write(1, red); //save red value to EEPROM
16      EEPROM.write(2, green); //save green value to EEPROM
17      EEPROM.write(3, blue); //save blue value to EEPROM
18      EEPROM.commit();
19      page = 1;  //->menu
20      line = 1;
21      break;
22     case 4:      //Musik
23      page = 1;  //->menu
24      line = 2;
25      break;
26     case 5:      //Sonstiges
27      page = 1;  //->menu
28      line = 3;
29      break;
30     case 6:      //WeckerMenu 1
31      //save alarm values to EEPROM
32      EEPROM.write(8, wecker1.active);
33      EEPROM.write(13, wecker1.light);
34      EEPROM.write(15, wecker1.track);
35      EEPROM.commit();
36      page = 2;  //->Wecker
37      line = 0;
38      break;
39     case 7:      //WeckerMenu 2
40      //save alarm values to EEPROM
41      EEPROM.write(16, wecker2.active);
```

```
42     EEPROM.write(21, wecker2.light);
43     EEPROM.write(23, wecker2.track);
44     EEPROM.commit();
45     page = 2;    //->Wecker
46     line = 1;
47     break;
48 case 8:        //WeckerMenu 3
49     //save alarm values to EEPROM
50     EEPROM.write(24, wecker3.active);
51     EEPROM.write(29, wecker3.light);
52     EEPROM.write(31, wecker3.track);
53     EEPROM.commit();
54     page = 2;    //->Wecker
55     line = 2;
56     break;
57 case 9:        //WeckerMenu 4
58     //save alarm values to EEPROM
59     EEPROM.write(32, wecker4.active);
60     EEPROM.write(37, wecker4.light);
61     EEPROM.write(39, wecker4.track);
62     EEPROM.commit();
63     page = 2;    //->Wecker
64     line = 3;
65     break;
66 case 10:       //WeckerTime 1
67     //save alarm values to EEPROM
68     EEPROM.write(9, wecker1.hour);
69     EEPROM.write(10, wecker1.minute);
70     EEPROM.write(11, wecker1.repeat);
71     EEPROM.write(12, wecker1.weekday);
72     EEPROM.write(40, arrayToInt(wecker1.weekdays)); //convert array to
int and save value to EEPROM
73     EEPROM.commit();
74     page = 6;    //->WeckerMenu 1
75     line = 1;
76     break;
77 case 11:       //WeckerTime 2
78     //save alarm values to EEPROM
79     EEPROM.write(17, wecker2.hour);
80     EEPROM.write(18, wecker2.minute);
81     EEPROM.write(19, wecker2.repeat);
82     EEPROM.write(20, wecker2.weekday);
83     EEPROM.write(41, arrayToInt(wecker2.weekdays)); //convert array to
int and save value to EEPROM
84     EEPROM.commit();
```

```
85     page = 7;    //->WeckerMenu 2
86     line = 1;
87     break;
88     case 12:      //WeckerTime 3
89     //save alarm values to EEPROM
90     EEPROM.write(25, wecker3.hour);
91     EEPROM.write(26, wecker3.minute);
92     EEPROM.write(27, wecker3.repeat);
93     EEPROM.write(28, wecker3.weekday);
94     EEPROM.write(42, arrayToInt(wecker3.weekdays)); //convert array to
int and save value to EEPROM
95     EEPROM.commit();
96     page = 8;    //->WeckerMenu 3
97     line = 1;
98     break;
99     case 13:      //WeckerTime 4
100    //save alarm values to EEPROM
101    EEPROM.write(33, wecker4.hour);
102    EEPROM.write(34, wecker4.minute);
103    EEPROM.write(35, wecker4.repeat);
104    EEPROM.write(36, wecker4.weekday);
105    EEPROM.write(43, arrayToInt(wecker4.weekdays)); //convert array to
int and save value to EEPROM
106    EEPROM.commit();
107    page = 9;    //->WeckerMenu 4
108    line = 1;
109    break;
110    case 14:      //TIME DATE SETUP
111    page = 5;    //->Sonstiges
112    line = 2;
113    break;
114    case 15:      //Uhrzeit einstellen
115    page = 14;   //->Sonstiges
116    line = 0;
117    break;
118    case 16:      //Datum einstellen
119    page = 14;   //->Sonstiges
120    line = 1;
121    break;
122    default: break;
123 }
124 }
```


A.4. light.ino

Listing A.4: light.ino

```
1 //set light of LED panel to passed over values
2 void setLight(uint8_t r, uint8_t g, uint8_t b) {
3   for (int i = 0; i < NUMPIXELS; i++) pixels.setPixelColor(i, pixels.
4     Color(r, g, b));
5   pixels.show();
6 }
7 //set light of LED panel to configured values
8 void setLight() {
9   if (licht) setLight(red, green, blue);
10  if (!licht) {
11    pixels.clear();
12    pixels.show();
13  }
14 }
```

A.5. ntp.ino

Listing A.5: ntp.ino

```
1 //get time via NTP; returns time in seconds
2 //code from: https://github.com/PaulStoffregen/Time/blob/master/examples
  /TimeNTP/TimeNTP.ino
3 time_t getNtpTime() {
4   IPAddress ntpServerIP; // NTP server's ip address
5
6   while (Udp.parsePacket() > 0) ; // discard any previously received
  packets
7   Serial.println("Transmit NTP Request");
8   // get a random server from the pool
9   WiFi.hostByName(ntpServerName, ntpServerIP);
10  Serial.print(ntpServerName);
11  Serial.print(": ");
12  Serial.println(ntpServerIP);
13  sendNTPpacket(ntpServerIP);
14  uint32_t beginWait = millis();
15  while (millis() - beginWait < 1500) {
16    int size = Udp.parsePacket();
17    if (size >= NTP_PACKET_SIZE) {
18      Serial.println("Receive NTP Response");
19      Udp.read(packetBuffer, NTP_PACKET_SIZE); // read packet into the
  buffer
20      unsigned long secsSince1900;
21      // convert four bytes starting at location 40 to a long integer
22      secsSince1900 = (unsigned long)packetBuffer[40] << 24;
23      secsSince1900 |= (unsigned long)packetBuffer[41] << 16;
24      secsSince1900 |= (unsigned long)packetBuffer[42] << 8;
25      secsSince1900 |= (unsigned long)packetBuffer[43];
26      return secsSince1900 - 2208988800UL + timeZone * SECS_PER_HOUR;
27    }
28  }
29  Serial.println("No NTP Response :-(");
30  return 0; // return 0 if unable to get the time
31 }
32
33 //send an NTP request to the time server at the given address
34 //code from: https://github.com/PaulStoffregen/Time/blob/master/examples
  /TimeNTP/TimeNTP.ino
35 void sendNTPpacket(IPAddress &address) {
36   // set all bytes in the buffer to 0
37   memset(packetBuffer, 0, NTP_PACKET_SIZE);
```

```
38 // Initialize values needed to form NTP request
39 // (see URL above for details on the packets)
40 packetBuffer[0] = 0b11100011; // LI, Version, Mode
41 packetBuffer[1] = 0; // Stratum, or type of clock
42 packetBuffer[2] = 6; // Polling Interval
43 packetBuffer[3] = 0xEC; // Peer Clock Precision
44 // 8 bytes of zero for Root Delay & Root Dispersion
45 packetBuffer[12] = 49;
46 packetBuffer[13] = 0x4E;
47 packetBuffer[14] = 49;
48 packetBuffer[15] = 52;
49 // all NTP fields have been given values, now
50 // you can send a packet requesting a timestamp:
51 Udp.beginPacket(address, 123); //NTP requests are to port 123
52 Udp.write(packetBuffer, NTP_PACKET_SIZE);
53 Udp.endPacket();
54 }
```

A.6. other.ino

Listing A.6: other.ino

```
1 //convert bool array[7] to uint8_t; useful for saving wecker.weekdays[]
  to EEPROM
2 uint8_t arrayToInt(boolean array[7]) {
3   uint8_t x = 0;
4   for (int i = 0; i < 7; i++) {
5     if (array[i]) x+= pow(2, i);
6   }
7   return x;
8 }
9
10 //convert uint8_t to bool array[7]; useful for reading wecker.weekdays[]
    from EEPROM
11 void intToArray(uint8_t x, boolean *array) {
12   for (int i = 0; i < 7; i++) {
13     array[i] = x % 2;
14     x /= 2;
15   }
16 }
```

A.7. selectEvent.ino

Listing A.7: selectEvent.ino

```
1 //event/action, if Select-Button is pressed
2 void selectEvent() {
3   //time and date
4   if (page == 0) {
5     page = 1;
6     line = 0;
7     return;
8   }
9
10  //menu
11  if (page == 1) {
12    switch(line) {
13      case 0: page = 2; break;
14      case 1: page = 3; break;
15      case 2: page = 4; break;
16      case 3: page = 5; break;
17      default: break;
18    }
19    line = 0;
20    return;
21  }
22
23  //alarm menu
24  if (page == 2) {
25    switch(line) {
26      case 0: page = 6; break;
27      case 1: page = 7; break;
28      case 2: page = 8; break;
29      case 3: page = 9; break;
30      default: break;
31    }
32    line = 0;
33    return;
34  }
35
36  //light
37  if (page == 3) {
38    switch(line) {
39      case 0:
40        licht = !licht;
41        break;
```

```
42     case 1:
43         if (red >= 250) red = 0;
44         else red += 10;
45         break;
46     case 2:
47         if (green >= 250) green = 0;
48         else green += 10;
49         break;
50     case 3:
51         if (blue >= 250) blue = 0;
52         else blue += 10;
53         break;
54     }
55     setLight();
56     return;
57 }
58
59 //music
60 if (page == 4) {
61     switch(line) {
62         case 0:
63             mp3Player.play(1); //Play the first mp3
64             break;
65         case 1:
66             mp3Player.pause(); //pause the mp3
67             break;
68         case 2:
69             mp3Player.next(); //Play next mp3
70             break;
71         case 3:
72             if (volume >= 30) volume = 0;
73             else volume += 5;
74             mp3Player.volume(volume); //Set volume. From 0 to 30
75             break;
76     }
77     return;
78 }
79
80 //Sosnstiges
81 if (page == 5) {
82     switch(line) {
83         case 0:
84             //turn LCD Backlight on/off
85             displayLicht = !displayLicht;
86             if(displayLicht) display.backlight();
```

```
87     if(!displayLicht) display.noBacklight();
88     break;
89     case 1:
90         //start wifi manager to connect with a different wifi
91         wifiManager();
92         break;
93     case 2:
94         page = 14;
95         line = 0;
96         break;
97     case 3:
98         //restart microcontroller
99         ESP.restart();
100        break;
101    default: break;
102    }
103    return;
104 }
105
106 //WeckerMenu 1
107 if (page == 6) {
108     switch(line) {
109         case 0:
110             wecker1.active = !wecker1.active;
111             break;
112         case 1:
113             page = 10;
114             line = 0;
115             break;
116         case 2:
117             if (wecker1.light >= 40) wecker1.light = 20;
118             else wecker1.light += 10;
119             break;
120         case 3:
121             if (wecker1.track >= mp3Player.readFileCounts()) wecker1.track =
122             1;
123             else {
124                 wecker1.track++;
125                 //mp3Player.play(wecker1.track);
126             }
127             break;
128         }
129     return;
130 }
```

```
131 //WeckerMenu 2
132 if (page == 7) {
133     switch(line) {
134         case 0:
135             wecker2.active = !wecker2.active;
136             break;
137         case 1:
138             page = 11;
139             line = 0;
140             break;
141         case 2:
142             if (wecker2.light >= 40) wecker2.light = 20;
143             else wecker2.light += 10;
144             break;
145         case 3:
146             if (wecker2.track >= mp3Player.readFileCounts()) wecker2.track =
147             1;
148             else {
149                 wecker2.track++;
150                 //mp3Player.play(wecker2.track);
151             }
152             break;
153     }
154     return;
155 }
156 //WeckerMenu 3
157 if (page == 8) {
158     switch(line) {
159         case 0:
160             wecker3.active = !wecker3.active;
161             break;
162         case 1:
163             page = 12;
164             line = 0;
165             break;
166         case 2:
167             if (wecker3.light >= 40) wecker3.light = 20;
168             else wecker3.light += 10;
169             break;
170         case 3:
171             if (wecker3.track >= mp3Player.readFileCounts()) wecker3.track =
172             1;
173             else {
174                 wecker3.track++;
```



```
174     }
175     break;
176 }
177 return;
178 }
179
180 //WeckerMenu 4
181 if (page == 9) {
182     switch(line) {
183         case 0:
184             wecker4.active = !wecker4.active;
185             break;
186         case 1:
187             page = 13;
188             line = 0;
189             break;
190         case 2:
191             if (wecker4.light >= 40) wecker4.light = 20;
192             else wecker4.light += 10;
193             break;
194         case 3:
195             if (wecker4.track >= mp3Player.readFileCounts()) wecker4.track =
196 1;
197             else {
198                 wecker4.track++;
199             }
200             break;
201     }
202     return;
203 }
204
205 //WeckerTime 1
206 if (page == 10) {
207     switch(line) {
208         case 0:
209             if (wecker1.hour >= 23) wecker1.hour = 0;
210             else wecker1.hour++;
211             break;
212         case 1:
213             if (wecker1.minute >= 59) wecker1.minute = 0;
214             else wecker1.minute++;
215             break;
216         case 2:
217             if (wecker1.repeat >= 4) wecker1.repeat = 1;
218             else {
```

```
218     wecker1.repeat++;
219 }
220 if (wecker1.repeat == 0 || wecker1.repeat == 2) {
221     wecker1.weekday = EEPROM.read(12);
222     if (wecker1.weekday == 0) wecker1.weekday = 2;
223     for (int i = 0; i < 7; i++) {
224         if (i == wecker1.weekday-1) wecker1.weekdays[i] = true;
225         else wecker1.weekdays[i] = false;
226     }
227 }
228 if (wecker1.repeat == 1) {
229     wecker1.weekday = 0;
230     for (int i = 0; i < 7; i++) wecker1.weekdays[i] = true;
231 }
232 if (wecker1.repeat == 3) {
233     wecker1.weekday = 0;
234     for (int i = 0; i < 7; i++) {
235         if (i == 0 || i == 6) wecker1.weekdays[i] = false;
236         else wecker1.weekdays[i] = true;
237     }
238 }
239 if (wecker1.repeat == 4) {
240     wecker1.weekday = 0;
241     for (int i = 0; i < 7; i++) {
242         if (i == 0 || i == 6) wecker1.weekdays[i] = true;
243         else wecker1.weekdays[i] = false;
244     }
245 }
246 break;
247 case 3:
248     if (wecker1.repeat == 0 || wecker1.repeat == 2) {
249         for (int i = 0; i < 7; i++) wecker1.weekdays[i] = false;
250         if (wecker1.weekday >= 7) wecker1.weekday = 1;
251         else {
252             wecker1.weekday++;
253         }
254         wecker1.weekdays[wecker1.weekday-1] = true;
255     }
256     break;
257 }
258 return;
259 }
260
261 //WeckerTime 2
262 if (page == 11) {
```

```
263     switch(line) {
264         case 0:
265             if (wecker2.hour >= 23) wecker2.hour = 0;
266             else wecker2.hour++;
267             break;
268         case 1:
269             if (wecker2.minute >= 59) wecker2.minute = 0;
270             else wecker2.minute++;
271             break;
272         case 2:
273             if (wecker2.repeat >= 4) wecker2.repeat = 1;
274             else {
275                 wecker2.repeat++;
276             }
277             if (wecker2.repeat == 0 || wecker2.repeat == 2) {
278                 wecker2.weekday = EEPROM.read(12);
279                 if (wecker2.weekday == 0) wecker2.weekday = 2;
280                 for (int i = 0; i < 7; i++) {
281                     if (i == wecker2.weekday - 1) wecker2.weekdays[i] = true;
282                     else wecker2.weekdays[i] = false;
283                 }
284             }
285             if (wecker2.repeat == 1) {
286                 wecker2.weekday = 0;
287                 for (int i = 0; i < 7; i++) wecker2.weekdays[i] = true;
288             }
289             if (wecker2.repeat == 3) {
290                 wecker2.weekday = 0;
291                 for (int i = 0; i < 7; i++) {
292                     if (i == 0 || i == 6) wecker2.weekdays[i] = false;
293                     else wecker2.weekdays[i] = true;
294                 }
295             }
296             if (wecker2.repeat == 4) {
297                 wecker2.weekday = 0;
298                 for (int i = 0; i < 7; i++) {
299                     if (i == 0 || i == 6) wecker2.weekdays[i] = true;
300                     else wecker2.weekdays[i] = false;
301                 }
302             }
303             break;
304         case 3:
305             if (wecker2.repeat == 0 || wecker2.repeat == 2) {
306                 for (int i = 0; i < 7; i++) wecker2.weekdays[i] = false;
307                 if (wecker2.weekday >= 7) wecker2.weekday = 1;
```

```
308     else {
309         wecker2.weekday++;
310     }
311     wecker2.weekdays[wecker2.weekday-1] = true;
312 }
313 break;
314 }
315 return;
316 }
317
318 //WeckerTime 3
319 if (page == 12) {
320     switch(line) {
321         case 0:
322             if (wecker3.hour >= 23) wecker3.hour = 0;
323             else wecker3.hour++;
324             break;
325         case 1:
326             if (wecker3.minute >= 59) wecker3.minute = 0;
327             else wecker3.minute++;
328             break;
329         case 2:
330             if (wecker3.repeat >= 4) wecker3.repeat = 1;
331             else {
332                 wecker3.repeat++;
333             }
334             if (wecker3.repeat == 0 || wecker3.repeat == 2) {
335                 wecker3.weekday = EEPROM.read(12);
336                 if (wecker3.weekday == 0) wecker3.weekday = 2;
337                 for (int i = 0; i < 7; i++) {
338                     if (i == wecker3.weekday-1) wecker3.weekdays[i] = true;
339                     else wecker3.weekdays[i] = false;
340                 }
341             }
342             if (wecker3.repeat == 1) {
343                 wecker3.weekday = 0;
344                 for (int i = 0; i < 7; i++) wecker3.weekdays[i] = true;
345             }
346             if (wecker3.repeat == 3) {
347                 wecker3.weekday = 0;
348                 for (int i = 0; i < 7; i++) {
349                     if (i == 0 || i == 6) wecker3.weekdays[i] = false;
350                     else wecker3.weekdays[i] = true;
351                 }
352             }

```

```
353     if (wecker3.repeat == 4) {
354         wecker3.weekday = 0;
355         for (int i = 0; i < 7; i++) {
356             if (i == 0 || i == 6) wecker2.weekdays[i] = true;
357             else wecker2.weekdays[i] = false;
358         }
359     }
360     break;
361 case 3:
362     if (wecker3.repeat == 0 || wecker3.repeat == 2) {
363         for (int i = 0; i < 7; i++) wecker3.weekdays[i] = false;
364         if (wecker3.weekday >= 7) wecker3.weekday = 1;
365         else {
366             wecker3.weekday++;
367         }
368         wecker3.weekdays[wecker3.weekday-1] = true;
369     }
370     break;
371 }
372 return;
373 }
374
375 //WeckerTime 4
376 if (page == 13) {
377     switch(line) {
378         case 0:
379             if (wecker4.hour >= 23) wecker4.hour = 0;
380             else wecker4.hour++;
381             break;
382         case 1:
383             if (wecker4.minute >= 59) wecker4.minute = 0;
384             else wecker4.minute++;
385             break;
386         case 2:
387             if (wecker4.repeat >= 4) wecker4.repeat = 1;
388             else {
389                 wecker4.repeat++;
390             }
391             if (wecker4.repeat == 0 || wecker4.repeat == 2) {
392                 wecker4.weekday = EEPROM.read(12);
393                 if (wecker4.weekday == 0) wecker4.weekday = 2;
394                 for (int i = 0; i < 7; i++) {
395                     if (i == wecker4.weekday-1) wecker4.weekdays[i] = true;
396                     else wecker4.weekdays[i] = false;
397                 }

```

```
398     }
399     if (wecker4.repeat == 1) {
400         wecker4.weekday = 0;
401         for (int i = 0; i < 7; i++) wecker4.weekdays[i] = true;
402     }
403     if (wecker4.repeat == 3) {
404         wecker3.weekday = 0;
405         for (int i = 0; i < 7; i++) {
406             if (i == 0 || i == 6) wecker4.weekdays[i] = false;
407             else wecker4.weekdays[i] = true;
408         }
409     }
410     if (wecker4.repeat == 4) {
411         wecker4.weekday = 0;
412         for (int i = 0; i < 7; i++) {
413             if (i == 0 || i == 6) wecker4.weekdays[i] = true;
414             else wecker4.weekdays[i] = false;
415         }
416     }
417     break;
418 case 3:
419     if (wecker4.repeat == 0 || wecker4.repeat == 2) {
420         for (int i = 0; i < 7; i++) wecker4.weekdays[i] = false;
421         if (wecker4.weekday >= 7) wecker4.weekday = 1;
422         else {
423             wecker4.weekday++;
424         }
425         wecker4.weekdays[wecker4.weekday-1] = true;
426     }
427     break;
428 }
429 return;
430 }
431
432 //DATE TIME SETUP
433 if (page == 14) {
434     switch(line) {
435         case 0:
436             page = 15;
437             line = 0;
438             break;
439         case 1:
440             page = 16;
441             line = 0;
442             break;
```

```
443     case 2:
444         setTime(getNtpTime());
445         page = 0;
446         line = 0;
447         break;
448     case 3:
449         if (timeZone >= 2) timeZone = 1;
450         else timeZone++;
451         setTime(getNtpTime());
452         EEPROM.write(4, timeZone);
453         EEPROM.commit();
454         break;
455     default: break;
456 }
457 return;
458 }
459
460 //Uhrzeit einstellen
461 if (page == 15) {
462     switch(line) {
463         case 0:
464             if (hrSetup >= 23) hrSetup = 0;
465             else hrSetup++;
466             break;
467         case 1:
468             if (minSetup >= 23) minSetup = 0;
469             else minSetup++;
470             break;
471         case 2:
472             setTime(hrSetup, minSetup, 0, day(), month(), year());
473             page = 14;    //->Sonstiges
474             line = 0;
475             default: break;
476     }
477     return;
478 }
479
480 //Datum einstellen
481 if (page == 16) {
482     switch(line) {
483         case 0:
484             if (daySetup >= 31) daySetup = 1;
485             else daySetup++;
486             break;
487         case 1:
```

```
488     if (monthSetup >= 22) monthSetup = 1;
489     else monthSetup++;
490     break;
491     case 2:
492     if (yrSetup >= 2030) yrSetup = 2021;
493     else yrSetup++;
494     break;
495     case 3:
496     setTime(hour(), minute(), second(), daySetup, monthSetup,
yrSetup);
497     page = 14;    //->Sonstiges
498     line = 1;
499     default: break;
500 }
501 return;
502 }
503 }
```


A.8. setup.ino

Listing A.8: setup.ino

```
1 //configure pins as input and enable the internal pull-up resistor
2 void input() {
3   pinMode(DOWN, INPUT_PULLUP);
4   pinMode(SELECT, INPUT_PULLUP);
5   pinMode(BACK, INPUT_PULLUP);
6   pinMode(UP, INPUT_PULLUP);
7   pinMode(ALARM, INPUT_PULLUP);
8 }
9
10 //configure pins as output
11 void output() {
12   pinMode(LED, OUTPUT);
13   pinMode(SCL, OUTPUT);
14   pinMode(SDA, OUTPUT);
15 }
16
17 //configure interrupts
18 void interrupt() {
19   attachInterrupt(DOWN, pressedDOWN, FALLING);
20   attachInterrupt(UP, pressedUP, FALLING);
21   attachInterrupt(SELECT, pressedSELECT, FALLING);
22   attachInterrupt(BACK, pressedBACK, FALLING);
23   attachInterrupt(ALARM, pressedALARM, FALLING);
24 }
25
26 //udp connection for ntp
27 void udp() {
28   Serial.println("Starting UDP...");
29   Udp.begin(localPort);
30   Serial.print("Local port: ");
31   Serial.println(localPort);
32 }
33
34 // initialize the lcd
35 void lcd() {
36   display.init();
37   display.backlight();
38   display.setCursor(0,0);
39   display.print("Starting clock...");
40 }
41
```

```
42 //connect to WiFi (uses wifi manager)
43 void wlan() {
44   WiFi.mode(WIFI_STA); // explicitly set mode, esp defaults to STA+AP
45   WiFiManager wm;
46
47   display.setCursor(0,1);
48   display.print("Bei neuem WLAN mit");
49   display.setCursor(1,2);
50   display.print("AP >Lichtwecker");
51   display.setCursor(1,3);
52   display.print("Setup< verbinden!");
53
54   //reset settings - wipe credentials for testing
55   //wm.resetSettings();
56
57   boolean res;
58   res = wm.autoConnect("Lichtwecker Setup");
59
60   if(!res) {
61     Serial.println("Failed to connect");
62     // ESP.restart();
63     page = 14;
64     line = 0;
65     prevPressed = !pressed;
66   }
67   else {
68     //if you get here you have connected to the WiFi
69     Serial.println("connected to wifi");
70   }
71 }
72
73 //on demand wifi manager (can be called from Sonstiges menu page)
74 void wifiManager() {
75   WiFiManager wm;
76
77   //reset settings - for testing
78   //wifiManager.resetSettings();
79
80   // set configportal timeout
81   wm.setConfigPortalTimeout(120);
82
83   if (!wm.startConfigPortal("Lichtwecker Setup")) {
84     Serial.println("failed to connect and hit timeout");
85     delay(3000);
86     ESP.restart();
```

```
87     delay(5000);
88 }
89
90 //if you get here you have connected to the WiFi
91 Serial.println("connected to wifi");
92 }
93
94 //initialize EEPROM
95 void flash() {
96     EEPROM.begin(EEPROM_SIZE);
97 }
98
99 //initial ntp time synch; set synch interval
100 void ntp() {
101     timeZone = EEPROM.read(4);
102     Serial.println("waiting for sync");
103     setSyncProvider(getNtpTime);
104     setSyncInterval(10800); //NTP Sync every 3 hours (time in seconds)
105 }
106
107 //setup mp3 player
108 void mp3() {
109     if (!mp3Player.begin(Serial2)) {
110         Serial.println(F("Unable to begin:"));
111         Serial.println(F("1.Please recheck the connection!"));
112         Serial.println(F("2.Please insert the SD card!"));
113     }
114     Serial.println(F("DFPlayer Mini online."));
115
116     mp3Player.volume(volume); //Set volume value. From 0 to 30
117 }
118
119 //read light values from EEPROM and set light
120 void light() {
121     licht = EEPROM.read(0);
122     red = EEPROM.read(1);
123     green = EEPROM.read(2);
124     blue = EEPROM.read(3);
125
126     setLight();
127 }
128
129 //read alarm clock values from EEPROM
130 void initWecker() {
131     wecker1.active = EEPROM.read(8);
```

```
132  wecker1.hour =EEPROM.read(9);
133  wecker1.minute =EEPROM.read(10);
134  wecker1.repeat =EEPROM.read(11);
135  wecker1.weekday =EEPROM.read(12);
136  wecker1.light =EEPROM.read(13);
137  // wecker1.volume =EEPROM.read(14);
138  wecker1.track =EEPROM.read(15);
139  intToArray(EEPROM.read(40), wecker1.weekdays);
140
141  wecker2.active = EEPROM.read(16);
142  wecker2.hour =EEPROM.read(17);
143  wecker2.minute =EEPROM.read(18);
144  wecker2.repeat =EEPROM.read(19);
145  wecker2.weekday =EEPROM.read(20);
146  wecker2.light =EEPROM.read(21);
147  // wecker2.volume =EEPROM.read(22);
148  wecker2.track =EEPROM.read(23);
149  intToArray(EEPROM.read(41), wecker2.weekdays);
150
151  wecker3.active = EEPROM.read(24);
152  wecker3.hour =EEPROM.read(25);
153  wecker3.minute =EEPROM.read(26);
154  wecker3.repeat =EEPROM.read(27);
155  wecker3.weekday =EEPROM.read(28);
156  wecker3.light =EEPROM.read(29);
157  // wecker3.volume =EEPROM.read(30);
158  wecker3.track =EEPROM.read(31);
159  intToArray(EEPROM.read(42), wecker3.weekdays);
160
161  wecker4.active = EEPROM.read(32);
162  wecker4.hour =EEPROM.read(33);
163  wecker4.minute =EEPROM.read(34);
164  wecker4.repeat =EEPROM.read(35);
165  wecker4.weekday =EEPROM.read(36);
166  wecker4.light =EEPROM.read(37);
167  // wecker4.volume =EEPROM.read(38);
168  wecker4.track =EEPROM.read(39);
169  intToArray(EEPROM.read(43), wecker4.weekdays);
170 }
```

A.9. writeDisplay.ino

Listing A.9: writeDisplay.ino

```
1 //methods to display menu on LCD
2
3 void printTimeSetup() {
4   display.setCursor(1,0);
5   display.print("Stunde: ");
6   printDigits(hrSetup);
7   display.setCursor(1,1);
8   display.print("Minute: ");
9   printDigits(minSetup);
10  display.setCursor(1,2);
11  display.print("SPEICHERN");
12  display.setCursor(0,line);
13  display.print(">");
14 }
15
16 void printDateSetup() {
17   display.setCursor(1,0);
18   display.print("Tag: ");
19   printDigits(daySetup);
20   display.setCursor(1,1);
21   display.print("Monat: ");
22   printDigits(monthSetup);
23   display.setCursor(1,2);
24   display.print("Jahr: ");
25   display.print(yrSetup);
26   display.setCursor(1,3);
27   display.print("SPEICHERN");
28   display.setCursor(0,line);
29   display.print(">");
30 }
31
32 void printTimeDateSetup() {
33   display.setCursor(1,0);
34   display.print("Uhrzeit einstellen");
35   display.setCursor(1,1);
36   display.print("Datum einstellen");
37   display.setCursor(1,2);
38   display.print("NTP Synchronisation");
39   display.setCursor(1,3);
40   display.print("Zeitzone: UTC+");
41   display.print(timeZone);
```

```
42  display.setCursor(0,line);
43  display.print(">");
44  }
45
46  void printWeckerTime(wecker wecker) {
47  display.setCursor(1,0);
48  display.print("Stunde: ");
49  printDigits(wecker.hour);
50  display.setCursor(1,1);
51  display.print("Minute: ");
52  printDigits(wecker.minute);
53  display.setCursor(1,2);
54  display.print("Wdh: ");
55  display.print(wiederholung[wecker.repeat]);
56  display.setCursor(1,3);
57  display.print("Wochentag: ");
58  display.print(wochentag[wecker.weekday]);
59  display.setCursor(0,line);
60  display.print(">");
61  }
62
63  void printWeckerMenu(wecker wecker) {
64  display.setCursor(1,0);
65  display.print("Status: ");
66  display.print(wecker.active ? "Ein" : "Aus");
67  display.setCursor(1,1);
68  display.print("Uhrzeit & Tag");
69  display.setCursor(1,2);
70  display.print("Lichtvorlauf: ");
71  display.print(wecker.light);
72  display.print("min");
73  display.setCursor(1,3);
74  display.print("Wecksound: ");
75  display.print(wecker.track);
76  display.setCursor(0,line);
77  display.print(">");
78  }
79
80  void printSonstiges() {
81  display.setCursor(1,0);
82  display.print("Display: ");
83  display.print(displayLicht ? "Ein" : "Aus");
84  display.setCursor(1,1);
85  display.print("Start WiFi Manager");
86  display.setCursor(1,2);
```

```
87  display.print("Zeit einstellen");
88  display.setCursor(1,3);
89  display.print("Neustarten");
90  display.setCursor(0,line);
91  display.print(">");
92 }
93
94 void printMusik() {
95  display.setCursor(1,0);
96  display.print("Play");
97  display.setCursor(1,1);
98  display.print("Stop");
99  display.setCursor(1,2);
100 display.print("Next");
101 display.setCursor(1,3);
102 display.print("Volume: ");
103 display.print(volume);
104 display.setCursor(0,line);
105 display.print(">");
106 }
107
108 void printLicht() {
109  display.setCursor(1,0);
110  display.print("Licht: ");
111  display.setCursor(8,0);
112  display.print(licht ? "Ein" : "Aus");
113  display.setCursor(1,1);
114  display.print("Rot: ");
115  display.setCursor(8,1);
116  display.print(red);
117  display.setCursor(1,2);
118  display.print("Gruen: ");
119  display.setCursor(8,2);
120  display.print(green);
121  display.setCursor(1,3);
122  display.print("Blau: ");
123  display.setCursor(8,3);
124  display.print(blue);
125  display.setCursor(0,line);
126  display.print(">");
127 }
128
129 void printWecker() {
130  display.setCursor(1,0);
131  display.print("Wecker 1 |");
```

```
132 display.print(wecker1.active ? "Ein|" : "Aus|");
133 printDigits(wecker1.hour);
134 display.print(":");
135 printDigits(wecker1.minute);
136 display.setCursor(1,1);
137 display.print("Wecker 2 |");
138 display.print(wecker2.active ? "Ein|" : "Aus|");
139 printDigits(wecker2.hour);
140 display.print(":");
141 printDigits(wecker2.minute);
142 display.setCursor(1,2);
143 display.print("Wecker 3 |");
144 display.print(wecker3.active ? "Ein|" : "Aus|");
145 printDigits(wecker3.hour);
146 display.print(":");
147 printDigits(wecker3.minute);
148 display.setCursor(1,3);
149 display.print("Wecker 4 |");
150 display.print(wecker4.active ? "Ein|" : "Aus|");
151 printDigits(wecker4.hour);
152 display.print(":");
153 printDigits(wecker4.minute);
154 display.setCursor(0,line);
155 display.print(">");
156 }
157
158 void printMenu() {
159 display.setCursor(1,0);
160 display.print("Wecker");
161 display.setCursor(1,1);
162 display.print("Licht");
163 display.setCursor(1,2);
164 display.print("Musik");
165 display.setCursor(1,3);
166 display.print("Sonstiges");
167 display.setCursor(0,line);
168 display.print(">");
169 }
170
171 void digitalClockDisplay() {
172 //time
173 display.setCursor(6,0);
174 printDigits(hour());
175 display.print(":");
176 printDigits(minute());
```



```
177   display.print(":");
178   printDigits(second());
179   //line
180   display.setCursor(0,1);
181   display.print("-----");
182   //date
183   display.setCursor(3,2);
184   display.print(wochentag[weekday()]);
185   display.print(", ");
186   printDigits(day());
187   display.print(".");
188   printDigits(month());
189   display.print(".");
190   display.print(year());
191 }
192
193 //utility for digital clock display; prints 0 if needed
194 void printDigits(int digits) {
195   if (digits < 10) display.print("0");
196   display.print(digits);
197 }
```

B. Anhang: Menüstruktur

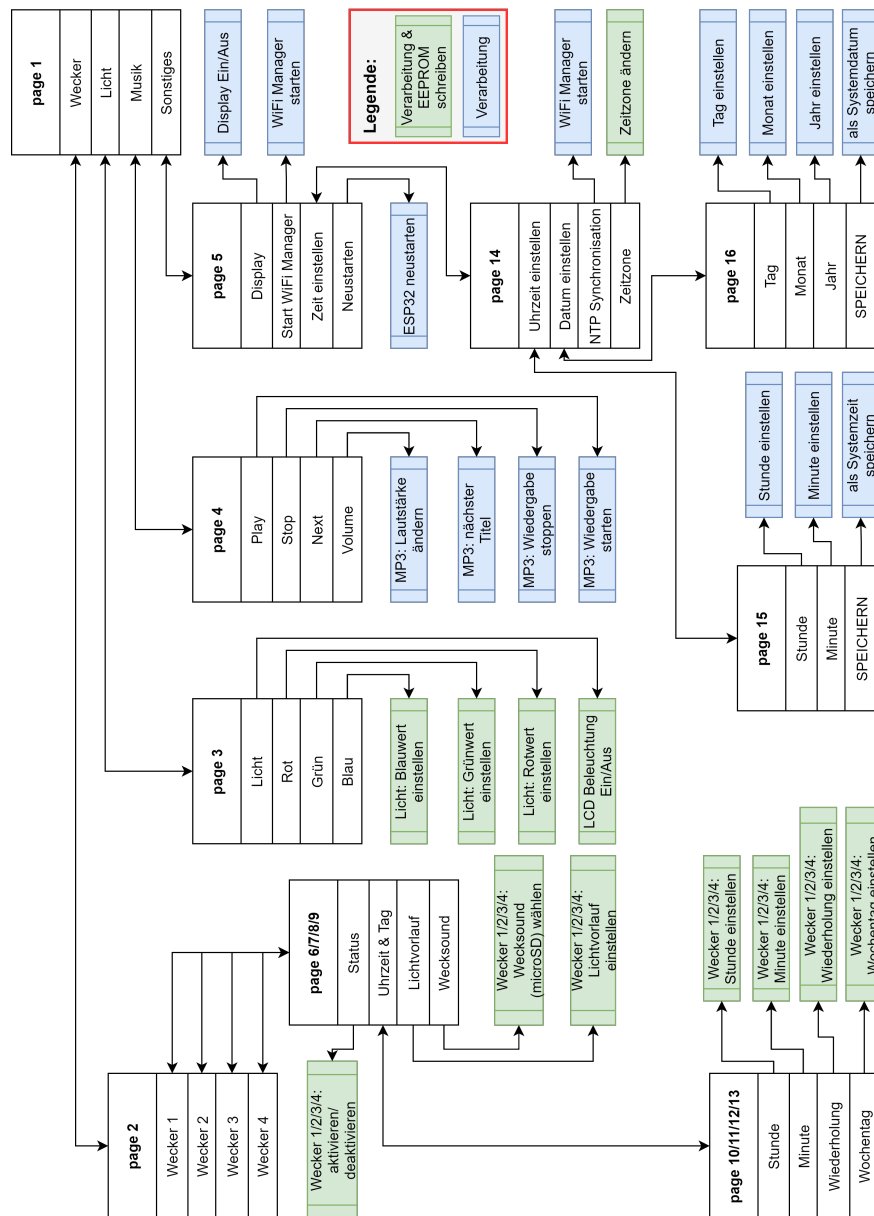


Abbildung B.1.: Menüstruktur

C. Anhang: loop()-Ablaufdiagramm

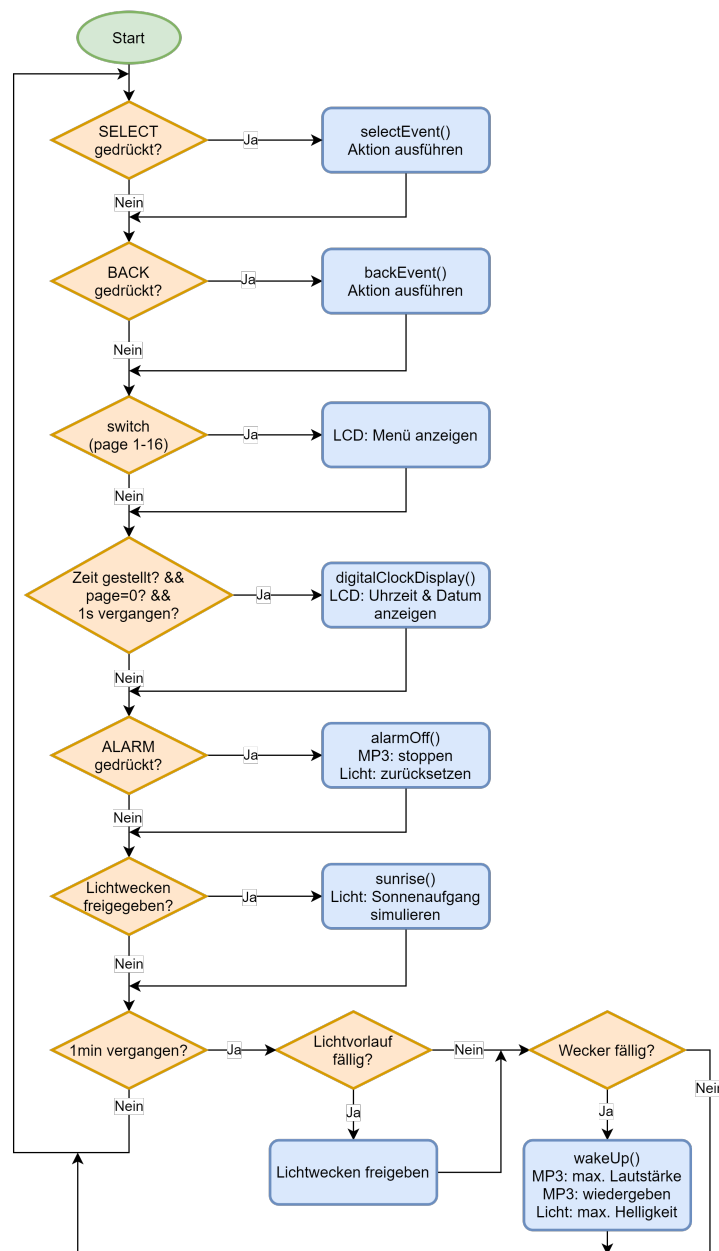


Abbildung C.1.: Ablaufdiagramm der loop()-Funktion

D. Anhang: digitaler Anhang

Der beigelegte Datenträger enthält folgende Dateien:

- 2021-03-27_V28 (Ordner mit Arduino Quellcode des Lichtweckers)
 - 2021-03-27_V28.ino
 - alarm.ino
 - backEvent.ino
 - light.ino
 - ntp.ino
 - other.ino
 - selectEvent.ino
 - setup.ino
 - writeDisplay.ino
- Messwerte_Ausfuehrungszeit_loop.xlsx (Ausführungszeit-Messwerte der loop() inklusive Klassifizierung und Histogramm)
- Studienarbeit_I_Schackniss_Johannes.pdf (schriftliche Ausarbeitung der Studienarbeit I)