

Besondere Lernleistung  
im Fach Informatik

# **Entwicklung, Bau und Programmierung eines computergesteuerten MP3-Weckers**

Liebigschule Frankfurt am Main

Eingereicht bei:  
Franziska Podesta

Vorgelegt von:  
Henri Wagner  
Fuchstanzstraße 29,  
60489 Frankfurt am Main  
Email: [wagner@frag-henri.de](mailto:wagner@frag-henri.de)

Abgabe: 21.03.2019

## Inhaltsverzeichnis

<b>1 Zielsetzung.....</b>	<b>1</b>
1.1 Anforderungen.....	1
1.1.1 Technische Anforderungen.....	1
1.1.2 Optische Anforderungen.....	2
1.2 Finanzialer Rahmen.....	2
<b>2 Hintergrund.....</b>	<b>3</b>
2.1 Produkte auf dem Markt.....	3
2.2 Technische Herausforderungen.....	4
2.2.1 Wahl der Hardwareplattform.....	4
2.2.2 Zeitvor- und -nachgang.....	5
2.2.3 Darstellung.....	6
2.2.4 Bedienelemente.....	7
2.2.5 Audiowiedergabe.....	8
2.3 Handwerkliche Herausforderungen.....	9
<b>3 Vorgehen.....</b>	<b>9</b>
3.1 Technische Methoden.....	9
3.1.1 Wahl der Hardwareplattform.....	9
3.1.2 Zeitvor- und -nachgang.....	10
3.1.3 Darstellung.....	12
3.1.4 Bedienelemente.....	12
3.1.5 Audiowiedergabe.....	16
3.2 Handwerkliche Methoden.....	16
3.3 Elektrotechnische Methoden.....	18
<b>4 Ergebnisse.....</b>	<b>19</b>
<b>5 Diskussion.....</b>	<b>30</b>
5.1 Reflektion der Herangehensweise.....	30
5.2 Reflektion von gewonnenen Erkenntnissen.....	31
5.3 Bewertung des Produktes.....	32

---

<b>6 Materialliste.....</b>	<b>33</b>
<b>7 Verzeichnisse.....</b>	<b>35</b>
Tabellenverzeichnis.....	35
Codeverzeichnis.....	35
Abbildungsverzeichnis.....	35
<b>8 Quellenangaben.....</b>	<b>36</b>
8.1 Abbildungsquellen.....	36
8.2 Literaturquellen.....	37

# 1 Zielsetzung

Zielsetzung war in erster Linie einen eigenständigen Wecker zu besitzen, welcher auf einem Nachttisch stehen kann. Bei Betrachtung des Markts fiel mir jedoch auf, dass es durchaus viele verschiedene Produkte gibt, die in einzelnen Punkten auch überzeugen können, im Gesamtbild jedoch immer mehrere Sachen vernachlässigt wurden. Der Wecker sollte in folgenden Punkten überzeugen:

- Bauqualität: hochwertige Materialien
- Äußere Erscheinung: schlichtes Design
- Funktionsumfang: eigene Songs in alternierender Reihenfolge, gedimmte Anzeige für die Nacht
- Genauigkeit: kein Nachstellen durch hohe Genauigkeit oder Funk
- Stromversorgung: Batterieloser Betrieb
- Datenschutz: keine Internetanbindung
- Kosten: bis 60€

Da die Produkte des Marktes mich jedoch in diesen Punkten nicht überzeugen konnten, wollte ich ein eigenes Produkt entwerfen und dies zu einem Projekt machen. Damit einhergehend erweiterte sich die Zielsetzung neben dem materiellen Besitz des Weckers noch um weitere Punkte. So erhoffte ich mir Einblicke in wirtschaftliche Abläufe zu bekommen. Dadurch, dass ich selbst die Prozedur der Entwicklung eines eigenen Produktes, sowohl auf Hard- als auch auf Softwareebene durchführte. Des Weiteren sollten sich meine Kenntnisse und Erfahrungen in diesen informationstechnischen Bereichen erweitern und unter Umständen weiter mein Interesse wecken.

Die Anforderungen, welche ich an mein Produkt stellte, möchte ich im Folgenden ausführen.

## 1.1 Anforderungen

### 1.1.1 Technische Anforderungen

Integraler Bestandteil des Weckers sollte von Anfang an die Möglichkeit sein, eigene Songs oder Töne anstelle eines monotonen Wecktons abspielen zu können.

Um Ausfällen vorzubeugen, wollte ich sowohl auf einen Akkumulator sowie auf komplexe Software verzichten, da durch eine einfache Software die Wahrscheinlichkeit für das Auftreten von Fehlern erheblich gesenkt wird.

Trotzdem sollte der Funktionsumfang des Weckers über den standardmäßigen hinausgehen und neben der Möglichkeit, eigene Songs abspielen zu können, auch Funktionen wie die Anzeige des

---

aktuellen Datums beinhalten. Des Weiteren wollte ich gerne das Problem des Uhrstellens – soweit es geht – umgehen. Die Uhr sollte entweder sehr genau takten und daher eine geringe Abweichung vorweisen oder sich automatisch selbst stellen, sobald eine gewisse Abweichung zu messen ist. Damit einhergehend sollte sie außerdem eine mögliche Zeitumstellung erkennen und diese durchführen. Diese Funktion hatte jedoch geringere Priorität, da die Abschaffung der Zeitumstellung im Raum steht.

Damit die digitale Anzeige des Weckers mich beim Einschlafen nicht behindert, wollte ich eine Möglichkeit haben, ihre Helligkeit zu variieren. Bei Nacht oder geringem Umgebungslicht sollte sich die Anzeige also wesentlich in der Helligkeit verändern, sodass sie noch immer einfach zu lesen ist, jedoch keine großen Mengen an Licht emittiert.

### **1.1.2 Optische Anforderungen**

Ein wichtiger Teil eines jeden Gegenstandes ist seine äußere Erscheinung. So sollte auch der Wecker nicht nur viele Funktionen erfüllen, sondern diese auch auf eine ansehnliche Art umsetzen. Ziel sollte ein schlichtes Design mit möglichst wenigen Bedienelementen sein, möglicherweise unter Verwendung natürlicher Materialien wie beispielsweise Holz. Neben der optischen Erscheinung musste der Wecker jedoch auch praktikabel sein. Auf filigrane Teile sollte aus Gründen der Bedienbarkeit und Langlebigkeit verzichtet werden, sodass auch in einem übermüdeten Zustand nichts zu Bruch ginge.

## **1.2 Finanzialer Rahmen**

Wichtiger Entscheidungsfaktor sind immer auch Kosten. Aufgrund der erweiterten Zielsetzung beim Eigenbau des Weckers bin ich in diesem Fall bereit, wesentlich mehr zu zahlen als für ein handelsübliches Produkt vom Markt. So hätte ich für einen fertigen Wecker nicht mehr als 60€ gezahlt, war jedoch bereit, für mein Projekt einen Rahmen von bis zu 120€ zu setzen, um viele Funktionen umsetzen zu können und daher auch mit meinem Endprodukt zufrieden zu sein.

Für solch einen Eigenbau ist diese Grenze jedoch trotzdem nicht sehr hoch. Schon bei Dingen wie den Baumaterialien liegen die Kosten aus dem Baumarkt höher als einfache handelsübliche Wecker kosten. Um die Kosten zu drücken, schaute ich nach Möglichkeiten, meine benötigten Teile zu einem geringem Preis einzukaufen und stellte schnell fest, dass Versandhäuser aus Fernost die exakt gleichen Teile für ein Bruchteil dessen, was lokale Händler verlangen, verkaufen. Da der Versand aus China zwar meistens kostenlos, jedoch nicht immer zollfrei und umweltfreundlich ist, versuchte ich, zum einen meine Bestellungen unterhalb von 26€ zu halten, um von zollamtlicher Behandlung befreit zu sein, gleichzeitig aber keine Einzelbestellungen zu tätigen, um der Umwelt nicht unnötig zu schaden.

## 2 Hintergrund

### 2.1 Produkte auf dem Markt

Im Folgenden Liste ich drei verschiedene Wecker auf, welche ich im Internet zu kaufen finden konnte. Alle haben gemeinsam, nicht unbedingt auf eine Batterie angewiesen zu sein und wurden jeweils mit mindestens dreieinhalf von fünf Sternen auf Amazon bewertet. Bei meiner Recherche fand ich jedoch keinen, welcher sich bei Zeitumstellung anpasst oder gar einen Funkempfänger besitzt. Des Weiteren scheint die Verwendung von Echtholz nicht sehr beliebt, lediglich Holzfurnier oder Acrylglass in Holzoptik wird verwendet.



*Abbildung 1:  
Produktbild  
Muse M-196<sup>I</sup>*

*Abbildung 2:  
Produktbild  
Pure Evoke H4<sup>II</sup>*

*Abbildung 3:  
Produktbild  
Floureon Wecker<sup>III</sup>*

<b>Materialien</b>	Kunststoff, Holzoptik	Kunststoff, Holzoptik	Kunststoff, Holzoptik
<b>Musikeingänge</b>	BT / AUX / UKW	BT / AUX / DAB	keine
<b>Weckfunktionen</b>	Radio	Radio	Weckton
<b>Anzeige</b>	7 Segment Display (dimmbar)	LC-Display (dimmbar)	7 Segment Display (dimmbar)
<b>Uhrzeitstellung</b>	Manuell	Manuell	Manuell
<b>Bedienung</b>	Kleine Knöpfe vorne, Großer Snooze Knopf oben	Kleine Knöpfe und große Drehknöpfe vorne, Snooze Touch Griff oben	Sehr kleine Knöpfe hinten
<b>Maße (H/B/T, mm)</b>	108 x 226 x 86	180 x 221 x 128	70 x 148 x 40
<b>Stromversorgung</b>	2xAAA / 230V	230V	3xAAA / 5V
<b>Preis</b>	49€	97€	23€

*Tabelle 1: Gegenüberstellung verschiedener Wecker*

## 2.2 Technische Herausforderungen

### 2.2.1 Wahl der Hardwareplattform

Zu den technischen Herausforderungen zählte zunächst einmal die Wahl einer geeigneten Hardwareplattform und die damit einhergehende Programmierung auf derselben.

Die eine Möglichkeit wäre, einen Raspberry Pi<sup>1</sup> oder einen vergleichbaren SBC<sup>2</sup> zu benutzen. Diese vereinen eine Vielzahl an Schnittstellen zusammen mit einem SoC<sup>3</sup> auf einer einzigen Platine und haben den Vorteil, dass ein modernes GNU/Linux<sup>4</sup> auf ihnen laufen kann, wodurch man Freiheiten, wie z.B. die Wahl der Programmiersprache, hat. Des Weiteren sind Funktionen wie W-LAN oder Bluetooth bei solchen SBC meistens schon integriert, falls man solche verwenden möchte. Mit einem großen und komplexen Betriebssystem kommen neben den vielen Funktionen jedoch auch viele potentielle Fehlerquellen hinzu, welche meinen Wecker wieder ein Stück unzuverlässiger und instabiler machen würden. Zudem sind sowohl die Kosten, als auch die Hitzeentwicklung in einem geschlossenen Gehäuse, bei solch einem SBC zu beachten.

Die eben genannten Schwachstellen weist ein Mikrocontroller<sup>5</sup>, welcher bereits kompilierten Maschinencode<sup>6</sup> ausführt, nicht auf. Dieser basiert im Vergleich zu einem SoC meistens auf einem Prozessor mit wesentlich einfacherer Architektur. Der primär damit einhergehende Nachteil ist die Programmierung, da ein Programm für den Mikrocontroller spezifisch kompiliert werden muss. Zudem bieten solche Controller lediglich eine große Anzahl an Lötkontakten, jedoch keine direkten Anschlüsse, wie z.B. für USB<sup>7</sup>.

Um die Vorteile beider Welten zu vereinen gibt es Firmen wie Arduino<sup>8</sup>, welche Entwicklungsboards auf einfacher Mikrocontrollerbasis entwickeln und verkaufen. Da diese nicht die Möglichkeit bieten, ein Betriebssystem wie Linux zu booten, bietet Arduino die Arduino IDE<sup>9</sup>, in welcher man in der Programmiersprache C++ programmiert und ein passender Compiler mitgeliefert wird, welcher diese in Maschinencode kompiliert. Die Platine kann daraufhin via USB an den PC angeschlossen werden, um den Code in ihren Speicher zu schreiben und bei Bedarf ihre serielle Ausgabe anzuzeigen. Nachteil einer solchen Plattform sind in erster Linie die begrenzten Ressourcen. Mehr Ressourcen bieten daher Entwicklungsboards anderer Firmen, wie beispielsweise der ESP32 der chinesischen Firma Espressif Systems, welches mit Hilfe einer Erweiterung ebenfalls mit der Arduino IDE kompatibel ist.

1 Eine Reihe von kostengünstigen Einplatinencomputern in Kreditkartengröße; von der britischen Raspberry Pi Foundation entwickelt

2 Steht für: Single-Board-Computer, zu deutsch: Einplatinencomputer

3 Steht für: System on a Chip; Halbleiterchip, welcher Prozessor, Speicher und Peripherie auf einem einzelnen Chip vereinen

4 Unix ähnliches Betriebssystem, aufbauend auf dem seit 1991 von Linus Torvalds entwickelten Linux Kernel

5 Teilweise als SoC bezeichnet, jedoch oftmals wesentlich schlechter ausgestattet und leistungsschwächer

6 Für Computer direkt ausführbarer Programmcode, meist von einem Compiler aus höheren Sprachen kompiliert

7 Steht für: Universal Serial Bus; weit verbreitetes serielles Bussystem zur Kommunikation mit externen Geräten

8 Entwickler der quelloffenen Arduino Entwicklungsboards und der quoelloffenen Arduino IDE

9 Steht für: Integrated Development Environment, zu deutsch: integrierte Entwicklungsumgebung

			
<i>Abbildung 4: Produktbild Raspberry Pi Zero W<sup>IV</sup></i>	<i>Abbildung 5: Produktbild Arduino Uno<sup>V</sup></i>	<i>Abbildung 6: Produktbild ESP32<sup>VI</sup></i>	
<b>CPU<sup>10</sup></b>	32-Bit / 1-GHz / ARM	8-Bit / 16-MHz / RISC	32-Bit / 240-MHz / MIPS
<b>RAM<sup>11</sup></b>	512 MiB	2 KiB	512 KiB
<b>ROM<sup>12</sup></b>	SD-Card	32 KiB	2-16 MiB
<b>WLAN/BT</b>	b/g/n / 4.1 BLE	- / -	b/g/n / 4.2 BLE
<b>Power</b>	5V	5V / 7-12V	3.3V
<b>Preis</b>	14€	7€	5€

Tabelle 2: Gegenüberstellung verschiedener Entwicklungsplattformen

## 2.2.2 Zeitvor- und -nachgang

Ein großes Problem bei Uhren ist das regelmäßige Notwendige Neustellen wegen des Vor- oder Nachgangs. Die meisten Uhren besitzen für diesen Zweck eine sogenannte Real Time Clock (RTC), welche versucht, einen möglichst gleichbleibenden Takt zu gewährleisten (klassischerweise auch als Quarzuhr bekannt). Allerdings ist hierbei das Problem, dass durch Stromverlust oder Wärmeausdehnung des Quarzes der Taktzyklus gestört oder gar unterbrochen werden kann<sup>VII</sup>. Zur Lösung dieses Problems gibt es spezielle RTC, welche neben dem Schwingquarz noch Temperaturfühler verbauen, um die gegebene Frequenz in Relation zu der aktuellen Temperatur zu setzen, um diese gegebenenfalls anzupassen. Des Weiteren ist es üblich, eine Knopfzelle anschließen zu können, welche bei Stromverlust die fortführende Taktung des Quarzes sicherstellt. Bei meiner Recherche stieß ich beispielsweise auf die DS3231, eine Real Time Clock der Firma Maxim Integrated. Der Quarz dieses Taktgebers schwingt mit einer Frequenz von 32.768Hz bei einer Abweichung von plusminus zwei Takten pro einer Million. Auf ein Jahr hochgerechnet entspricht dies einer durchschnittlichen Abweichung von unter einer Minute. Bei Verwendung einer wiederaufladbaren Lithium Knopfzelle wäre zudem eine redundante Stromversorgung und damit ein Weiterzählen der Uhr gegeben. Die Kommunikation würde über den weit verbreiteten I<sup>2</sup>C<sup>13</sup> Datenbus erfolgen<sup>VIII</sup>.

10 Steht für: **C**entral **P**rocessing **U**nit, zu deutsch: Prozessor

11 Steht für: **R**andom **A**ccess **M**emory, zu deutsch: Arbeitsspeicher

12 Steht für: **R**ead **O**nly **M**emory, zu deutsch: Speicher, der nur gelesen wird

13 Steht für: **I**nter-**I**ntegrated **C**ircuit; ein weit verbreiteter serieller Datenbus

Neben dieser klassischen Implementierung, welche eine möglichst geringe Abweichung über einen gewissen Zeitraum versucht umzusetzen, gibt es noch die Möglichkeit, die Uhrzeit mit einer Quelle zu synchronisieren. Moderne Computersysteme fragen daher die aktuelle Zeit über das Internet ab, was ein W-LAN-Netzwerk in Reichweite und eine entsprechende Schnittstelle voraussetzt. Da solche Netzwerke jedoch oftmals nicht sehr stabil sind und leicht gestört werden, würde sich dies ebenfalls auf die Zuverlässigkeit des Weckers auswirken.

Des Weiteren wäre die Verwendung des DCF77<sup>14</sup>-Signals, wie es klassische Funkuhren tun, denkbar. Der Sender hierfür steht in Frankfurt und überträgt mit 77,5kHz die aktuelle Uhrzeit einer Atomuhr<sup>IX</sup>. Das größte Problem hierbei wäre jedoch die Unterbringung der etwa sechs Zentimeter langen Antenne. Bei Stromverlust müsste zudem die Uhrzeit neu empfangen werden.

Weitere Methoden, wie die Abfrage über GPS oder das Mobilfunknetz, wären theoretisch denkbar, wurden jedoch wegen ihrer Komplexität und des Kostenfaktors frühzeitig ausgeschlossen.

## 2.2.3 Darstellung

Da eine analoge Darstellung der Uhrzeit durch die gegebene Informationsverarbeitung nicht praktikabel ist, vergleiche ich hier nur verschiedene Möglichkeiten der digitalen Darstellung.

Zur Wahl stehende Displays wären zum einen einfache Siebensegmentanzeigen, welche man typischer Weise in Mikrowellen oder Öfen findet. Diese zeigen mit Hilfe von sieben einzeln ansteuerbaren Segmenten eine Ziffer an. Um eine vierziffrige Zahl darzustellen, wird jede Ziffer für einen Bruchteil einer Sekunde erleuchtet. Damit können die vier Ziffern durch lediglich sieben Leitungen für die Segmente und jeweils eine pro Ziffer angesteuert werden. Da bei mehreren Displays jedoch mehr Leitungen benötigt werden würden als eine gängige Entwicklungsplattform bietet, gibt es die Siebensegmentanzeigen oftmals mit integriertem Controller, welcher seriell mit der Mutterplatine kommuniziert und die Anbindung der einzelnen Leitungen übernimmt. Durch eine hohe Interpolationsrate des Controllers wird dem menschlichen Auge suggeriert, alle Ziffern würden gleichzeitig leuchten<sup>X</sup>.

Bei Verwendung solcher Siebensegmentanzeigen gäbe es verschiedene Kombinationsmöglichkeiten für die Anzeige der unterschiedlichen Informationen. So könnte man alle Anzeigen in einer vereinen, welche in einem gewissen Zeitabstand oder bei Verwendung der Bedienelemente für die Weckzeit alternierend ihre Informationen darstellt. Alternativ kann man die Darstellung auch voneinander trennen, sodass sowohl das Datum als auch die Weckzeit jeweils ein eigenes Display benutzen oder sich eines teilen, die Uhrzeit in jedem Fall jedoch eine eigene Anzeige verwendet.

Neben den klassischen Siebensegmentanzeigen gibt es noch OLED-Displays<sup>15</sup>, welche als neue Technologie noch keinen großen Platz im Alltag eingenommen haben. Neben der Verwendung in einigen Smartphones wird diese Technologie in manchen hochpreisigen Fernsehern verwendet.

14 Rufzeichen (oder Stationskennung) der Funkstelle in Frankfurt

15 Steht für: **O**rganic **L**ight **E**mitting **D**iode, zu deutsch: organische Leuchtdiode

Dementsprechend befinden sich diese Anzeigen ebenfalls in einem etwas höheren Preissegment, wobei sich kleinere monochrome Ausführungen noch im finanziellen Rahmen halten. Der Vorteil – im Vergleich zu den Siebensegmentanzeigen – ist hierbei in erster Linie der Aufbau. Anstelle einzeln beleuchtbarer Segmente sind es Pixel, welche die Möglichkeit bieten, komplexere Symbole oder Animationen darzustellen. Des Weiteren sind OLED-Displays um einiges dünner als Siebensegmentanzeigen, was sich im Zusammenbau positiv auswirken könnte. Die Nachteile sind neben dem erhöhten Preis jedoch auch eine verringerte Leuchtkraft, da eine große LED eines Segments wesentlich mehr Licht emittieren kann als eine OLED Pixelmatrix.

Bei Verwendung eines solchen OLED-Displays wäre die Kombination mit einer großen Siebensegmentanzeige oder unter Umständen die Verwendung mehrerer OLED-Displays, um eine größere Anzeigefläche zu bilden, denkbar. Die Kombinations- und Programmierungsmöglichkeiten wären vielseitig.

## 2.2.4 Bedienelemente

Nach dem erarbeiteten Konzept sollten die Bedienelemente intuitiv zu benutzen und ästhetisch ansprechend sein. So zog ich Taster zur Einstellung der Weckzeit gar nicht erst in Erwägung. Stattdessen standen verschiedene Implementierungsvarianten für Drehknöpfe zur Auswahl. Dort könnte entweder jeweils ein Drehknopf für die Stunden und einer für die Minuten zuständig sein oder man verwendet alternativ nur einen einzigen, welcher mit Hilfe von beispielsweise einem Tastendruck zwischen Minuten und Stunden wechselt.

Potentiometer wären eine Möglichkeit, Drehknöpfe zu implementieren. Diese lassen sich linear drehen und haben einen Start- und Endpunkt, so dass man sie nicht unendlich weiter drehen kann. Der eingestellte Widerstand des Potentiometers, welcher mit Hilfe eines Analog-Digital-Umsetzers des Mikrocontrollers gemessen wird, würde in Relation zu einer Weckzeit stehen. Die Implementierung sowie Beschaffung würde sich verhältnismäßig einfach gestalten<sup>XI</sup>.

In der Benutzung sehr ähnlich, im Aufbau jedoch etwas anders, wären Drehgeber oder rotary encoder im Englischen. Diese haben eine feste Anzahl an Positionen, die sie einnehmen können und geben mit Hilfe zweier Leitungen zur Kommunikation ihre Position relativ zur Letzten an. Hier würde man einen Schritt einer bestimmten Anzahl an Minuten oder Stunden zuordnen. Die absolute Position kann jedoch, anders als bei einem Potentiometer, nicht bestimmt werden, sondern muss in Software umgesetzt werden. Damit ist bei Stromverlust die Information über die aktuelle Position aber auch nicht mehr zugänglich<sup>XII</sup>.

Neben der Einstellung der Weckzeit benötigt man jedoch auch noch eine Möglichkeit, den Wecker für die nächste Nacht einzuschalten, bzw. ihn daraufhin auszuschalten, sobald man geweckt wurde. Diese beiden Funktionen kann man entweder in einem Taster oder Schalter vereinen oder man trennt sie in zwei Einzelne.

---

Bei einer Separierung beider Funktionen wäre es denkbar für das Ausschalten des Wecktons einen besonders großen Taster zu verwenden, welchen man blind vom Bett aus erreichen kann. Für diesen Zweck kämen beispielsweise klassische Notausschalter in Frage, welche wegen ihres klassischen Einsatzgebietes zudem auch besonders robust sind. Damit einhergehender Nachteil sind allerdings die erhöhten Kosten, welche sich oftmals im unteren bis mittleren zweistelligen Eurobereich befinden.

Alternativ wäre es denkbar, anstelle eines sichtbaren Tasters, eine Berührungserkennung in die Oberfläche zu integrieren. Diese hätte nicht nur den Vorteil optisch nicht sichtbar zu sein und damit gut dem angestrebten Designkonzept zu entsprechen, sondern wäre auch eine besonders billige Art der Implementierung. Zu beachten wäre jedoch, dass der Erfolg einer solchen Implementierung sehr von der Geschwindigkeit des verwendeten Mikroprozessors abhängt und zudem vermutlich relativ störungsanfällig ist. Des Weiteren wäre dies mit einem erheblich erhöhten Programmierungsaufwand verbunden<sup>XIII</sup>.

Um die Weckfunktion am Abend vorher zu aktivieren, könnte ein einfacher Schalter dienen, welcher durch seine Funktionsweise nicht nur einfach zu bedienen, sondern auch einfach abzulesen ist. Bei Verwendung eines Tasters müsste es eine zusätzliche Rückmeldung in Form einer LED oder des Ausschaltens eines Displays geben, damit man den aktuellen Zustand (Wecker ein-/ausgeschaltet) ablesen kann.

Bei Verbindung beider Funktionen in ein einzelnes Bauteil wäre ein Taster jedoch etwas intuitiver zu benutzen als ein Schalter.

## 2.2.5 Audiomeldung

Um wecken zu können, muss es eine Möglichkeit geben, Audio wiederzugeben. Bei einem Einplatinencomputer mit integriertem SoC wäre dies sehr einfach umzusetzen, da diese oftmals schon einen DAC<sup>16</sup> integriert haben und die Möglichkeit bieten, Audio Dateien auf einer SD Karte abzulegen. Es müsste lediglich ein Lautsprecher an den Audioausgang angeschlossen. Unter Umständen wäre es erforderlich, einen Verstärker dazwischen zu schalten.

Mikrocontroller wie der Arduino Uno oder der ESP32 hingegen bieten nicht die Möglichkeit, von einer SD Karte Dateien zu lesen und ihr interner Speicher besitzt kein Dateisystem und ist zudem auch viel zu klein für Audiodateien. Abhilfe schaffen extra Module, wie beispielsweise der DF-Player Mini, welcher einen Steckplatz für eine Mikro SD Karte besitzt, MP3<sup>17</sup> Dateien von dieser dekodieren kann und über einen integrierten Verstärker bereits Lautsprecher bis zu drei Watt betreiben kann. Eine serielle Kommunikation kann über die UART-Schnittstelle<sup>18</sup> zur Mutterplatine hergestellt werden und bietet somit alle Funktionen, die ein SBC auch bieten würde<sup>XIV</sup>.

---

16 Steht für: Digital Analog Converter, zu deutsch: Digital-Analog-Wandler

17 Verlustbehaftetes Komprimierungsverfahren für Audiodateien

18 Steht für: Universal Asynchronous Receiver Transmitter; Schnittstelle für asynchronen seriellen Datentransfer

## 2.3 Handwerkliche Herausforderungen

Um die von mir beschriebenen Anforderungen zu erfüllen, muss der Wecker möglichst schlicht und optisch ansprechend sein, gleichzeitig aber genügend Platz für die Hardware im Innenraum bieten.

Des Weiteren wäre es, bei Verwendung von Siebensegmentanzeigen, wünschenswert, diese hinter einer Holzfurnierschicht zu verstecken, sodass sie im ausgeschalteten Zustand nicht zu erkennen sind und auch im eingeschalteten Zustand eine schöne Illusion erzeugen. Um dies zu realisieren, ist einmal die Verwendung von Furnier aus echtem Holz denkbar, wobei der Preis hierfür verhältnismäßig hoch und die Lichtdurchlässigkeit begrenzt ist. Alternativ kann man Furnier aus Vinyl in Holzoptik verwenden, welches eine homogene Struktur vorweist und dadurch auch eher lichtdurchlässig ist. Nachteil hierbei wäre unter Umständen, dass es nicht wirklich wie Holz aussieht und sich vermutlich auch nicht so anfühlt, was man ebenfalls testen müsste.

Bei Verwendung von OLED-Displays wäre ihre Leuchtkraft vermutlich zu gering, um sie hinter Holzfurnier zu verstecken, weshalb ich versuchen würde, sie in einer Ebene mit der Holzfront zu legen und alle Platinenteile so gut es geht zu verstecken.

Ferner stellt der allgemeine Bau eines stabilen und genau passenden Gehäuses schon eine Herausforderung dar, da kleine Messunterschiede oder Abweichungen beim Sägen der Materialien große Auswirkungen auf die endgültige Stabilität und das Aussehen haben. Daher wäre die Verwendung eines kleinen Grundgerüstes, an welchem die Seitenteile befestigt werden, eine Möglichkeit, für verbesserte Stabilität zu sorgen. Das Gerüst könnte aus schmalen Kanthölzern gebaut werden, wobei die Seitenteile entweder aus MDF-Platten<sup>19</sup>, sofern Furnier benutzt wird, bestehen könnten; bzw. alternativ aus dünnen Profilplatten, welche eine optisch ansprechende Struktur aufweisen.

Um das Ganze wartungsfreundlich zu machen, wäre es vorteilhaft, einige Seitenteile abnehmen oder öffnen, und die Hardware herausnehmen zu können. Dadurch wäre es im Falle eines Defekts möglich, gewisse Teile einfach auszutauschen.

## 3 Vorgehen

### 3.1 Technische Methoden

#### 3.1.1 Wahl der Hardwareplattform

Für die Wahl der Hardwareplattform fielen Einplatinencomputer wie der Raspberry Pi schnell heraus, da diese eine wesentlich erhöhte Hitzeentwicklung, einen erhöhten Stromverbrauch und – aufgrund der komplexeren Software – eine größere Instabilität aufweisen.

<sup>19</sup> Steht für: mitteldichte Holzfaserplatte; feine, homogene Holzfaserplatte

Daraufhin zog ich nur den Arduino Uno in Erwägung, dessen hauptsächlicher Vorteil seine hervorragende Dokumentation darstellt und den ESP32, welcher nicht ganz so gut dokumentiert ist und weniger Bibliotheken<sup>20</sup> bereitstellt, dafür ein Vielfaches der Rechenkapazität bei einem ähnlichen Preis bietet. Mir stellte sich jedoch die Frage, ob ich das Mehr an Rechenleistung überhaupt benötigen würde, da die grundsätzliche Funktionsweise eines Weckers doch sehr simpel ist. Des Weiteren überlegte ich mir, ob ich das Gerät überhaupt am Internet betreiben wollen würde, da gerade im IoT<sup>21</sup> Bereich solche Plattformen gerne für fremde Zwecke missbraucht werden.

Schließlich entschied ich mich für die Verwendung eines Arduino Unos, da ich durch mangelnde Kenntnisse in der Mikroelektronik und Programmierung sehr gerne auf die große Menge an Dokumentationen und Hilfestellungen zurückgreifen können wollte.

Mit der Entscheidung für den Arduino Uno als Hardwareplattform stand auch die Softwareplattform fest. Entweder in der eigenen IDE oder in diversen Editoren von Drittherstellern kann in C++ programmiert werden, woraufhin dieser Code in Maschinencode kompiliert wird und mit Hilfe eines USB Kabels in den Speicher des Mikrocontrollers geschrieben wird. Jedes Programm, welches für einen Arduino geschrieben ist, besitzt normalerweise mindestens zwei Methoden<sup>22</sup>, `setup()` und `loop()`. Code, welcher in `setup()` steht, wird ausschließlich bei Programmstart ausgeführt, während sich `loop()` am Ende eines Durchlaufs, selbst erneut aufruft und sich dadurch ununterbrochen wiederholt. Durch Bibliotheken, welche zu Beginn eingebunden werden können, kann man die Standardmethoden von C++ um viele erweitern, welche zu einem großen Teil von Arduino selbst zur Verfügung gestellt werden. Damit kann beispielsweise die Konfiguration der einzelnen Pins oder eine serielle Kommunikation zum PC mit Hilfe eines einfachen Methodenaufrufs durchgeführt bzw. eingerichtet werden.

### 3.1.2 Zeitvor- und -nachgang

Da ich mich bereits gegen die Verwendung einer internetfähigen Plattform entschieden hatte, blieb mir noch die Wahl zwischen der Verwendung des DCF77 Funksignals oder einer RTC übrig. Obwohl das Frankfurter Funksignal der Real Time Clock gegenüber viele Vorteile vorweisen kann, entschied ich mich schließlich gegen die Verwendung derselben und für die Verwendung der DS3231. Bei einer jährlichen Abweichung von unter einer Minute würde ich das seltene Neustellen in Kauf nehmen, um dafür eine vereinfachte Implementierung zu haben, da die Dokumentation über entsprechende Funkempfänger nicht so ausführlich ist, wie die der Real Time Clocks.

Um die Zeit am Wecker ganz genau einzustellen, fügte ich eine Funktion hinzu, welche eine Texteingabe über die serielle Schnittstelle annimmt und damit die Real Time Clock stellt. Der einzugebende Text muss ein großes „T“ am Anfang besitzen, gefolgt von der aktuellen Zeit im

20 Ansammlungen an fertigen Funktionen und Methoden, auf welche zugegriffen werden kann

21 Steht für: Internet of Things, zu deutsch: Internet der Dinge

22 Objektorientierte Unterprogramme, in Form von Funktionen oder Prozeduren

UNIX Format<sup>23</sup>. Damit kann man von einem Linux Rechner aus sehr komfortabel die aktuelle Uhrzeit weitergeben, indem man zunächst mit Hilfe von stty<sup>24</sup> im Linux Terminal<sup>25</sup> einmalig Einstellungen für das neue Terminal-Gerät des Arduinos vornimmt und daraufhin die aktuelle Zeit im UNIX Format abfragt, welche man dann an das Terminal-Gerät weiterleitet. Zusammengefasst sieht es dann folgendermaßen aus<sup>xv</sup>:

```
$ stty -F /dev/ttyACM0 cs8 115200 ignbrk -brkint -imaxbel -opost -onlcr -isig  
-icanon -iexten -echo -echoe -echok -echoctl -echoke noflsh -ixon -crtsccts
```

*Codebeispiel 1: Einmaliges Einrichten via stty des Arduino Terminals, eingehängt an /dev/ttyACM0 mit einer Abtastrate von 115200 Hz*

```
$ date +T%s > /dev/ttyACM0
```

*Codebeispiel 2: Weiterleitung der Ausgabe von date an das Terminal des Arduinos (/dev/ttyACM0)*

Mit der Entscheidung für die Verwendung einer Real Time Clock musste die Sommer- und Winterzeit in Software implementiert werden. Für die Umstellung auf die Mitteleuropäische Sommerzeit (MESZ) ist der letzte Sonntag im März um zwei Uhr nachts festgelegt. Auf die Winterzeit (MEZ) wird am letzten Sonntag im Oktober um drei Uhr nachts umgestellt. Mit diesen Informationen kann mit Hilfe einiger Bedingungsabfragen festgestellt werden, ob die Zeit umgestellt werden muss. Diese Implementierungsarbeit kann einem jedoch von Bibliotheken abgenommen werden, wie ich sie verwendet habe. Bei der `Timezone.h` werden die eben genannten Bedingungen angegeben und diese rechnet die UTC Zeit dem entsprechend in die lokal gültige Zeit um.

```
1 //Daylight saving time via "Timezone.h"  
2 //creating the statements  
3 TimeChangeRule myDST = {"MESZ", Last, Sun, Mar, 2, +120};  
4 TimeChangeRule mySDT = {"MEZ", Last, Sun, Oct, 3, +60};  
5 //initializing an object of class "Timezone" with the attributes from above  
6 Timezone DE(myDST, mySDT);  
7 //giving the pointer to tcr to TimeChangeRule  
8 TimeChangeRule *tcr;  
9 //initializing utc and local time of type time_t  
10 time_t utc, local;
```

*Codebeispiel 3: Implementierung der Zeitumstellung*

23 Darstellung der aktuellen Zeit als vergangene Sekunden seit dem 01.01.1970 00:00 UTC in einer 32-Bit Zahl

24 Steht für: set teletype; UNIX Programm um Terminal Einstellungen zu setzen

25 Text Eingabe- und Ausgabeumgebung

### 3.1.3 Darstellung

Zur Darstellung der Informationen entschied ich mich gegen die Verwendung von OLED-Displays, da diese durch ihre geringere Leuchtkraft schlechter durch Materialien hindurch scheinen können und zudem einen erhöhten Kostenfaktor darstellen würden.

Stattdessen verwendete ich drei einzelne Siebensegmentanzeigen, welche ihre Informationen statisch und ohne Animationen anzeigen. Dadurch sollte sowohl die Benutzerfreundlichkeit erhöht werden, als auch der Ablenkungsfaktor verringert werden, da es meinem Designkonzept nach schlicht und unauffällig bleiben sollte. Damit die Uhrzeit jedoch einfach von Datum und Weckzeit zu unterscheiden ist, wird diese auf einem besonders großen Display dargestellt. Das dafür verwendete Modul besitzt eine Ziffernhöhe von drei Zentimetern und bietet die Kommunikation über den bereits erwähnten I<sup>2</sup>C Bus an. Die beiden anderen verwendeten Displays sind mit einer Ziffernhöhe 1,4 und 0,9 Zentimetern wesentlich kleiner und wurden jeweils händisch auf eine Platine mit einem TM1637 Controller gelötet, welcher die Kommunikation über ein unspezifiziertes serielles Protokoll ermöglicht, wofür es jedoch eine einfach zu verwendende Bibliothek gibt.

Des Weiteren sorgte ich dafür, dass die Displays für Datum und Weckzeit zwischen einundzwanzig Uhr abends und elf Uhr morgens ausgeschaltet sind und das Display für die Uhrzeit bei minimaler Helligkeit leuchtet. Bei Berührung schalten sich diese mit verringelter Helligkeit jedoch an, während sie während des Klingelns mit maximaler Helligkeit leuchten.

### 3.1.4 Bedienelemente

Für die Bedienung entschied ich mich für die Verwendung zweier Drehgeber, wovon einer die Stunde und einer die Minute, zu der geweckt werden soll, einstellt. Ich entschied mich gegen die Verwendung von Potentiometern, da ich lieber einen kontinuierlich drehbaren Regler haben wollte, bei welchem zudem durch mehrere Umdrehungen die Zeit genauer einzustellen wäre. Ferner hätte ich die Bedienung mit lediglich einem Regler verwirklichen können. Aus Gründen der einfacheren Bedienbar- und Implementierbarkeit entschied ich mich jedoch für die Verwendung zweier Rotary Encoder.

Die verwendeten Encoder mit der Bezeichnung „KY-040“ besitzen insgesamt fünf Kontakte. Ihr Aufbau ist nahezu rein mechanisch; lediglich zwei Widerstände wurden verbaut. Bei Drehung würden die Pins A und B mit einer gewissen Zeitverzögerung von HIGH auf LOW oder umgekehrt gezogen. Wenn vom Mikrocontroller eine Veränderung im Zustand von Pin A festgestellt wird, kann Pin B validiert werden. Wenn dieser nun den gleichen Zustand wie Pin A besitzt, kann daraus geschlussfolgert werden, dass er zuerst seinen Zustand veränderte. Wenn Pin B sich jedoch nicht im gleichen Zustand wie Pin A befindet, kann daraus geschlussfolgert werden, dass sich Pin A zuerst veränderte. Dadurch wird die Richtung bestimmt, in welche sich der Encoder dreht. Um die Kontakte jeweils auf HIGH oder LOW legen zu können, besitzt der Encoder noch zwei weitere Kontakte für die Verbindung zu einer Stromquelle und für die Verbindung zur Erde.

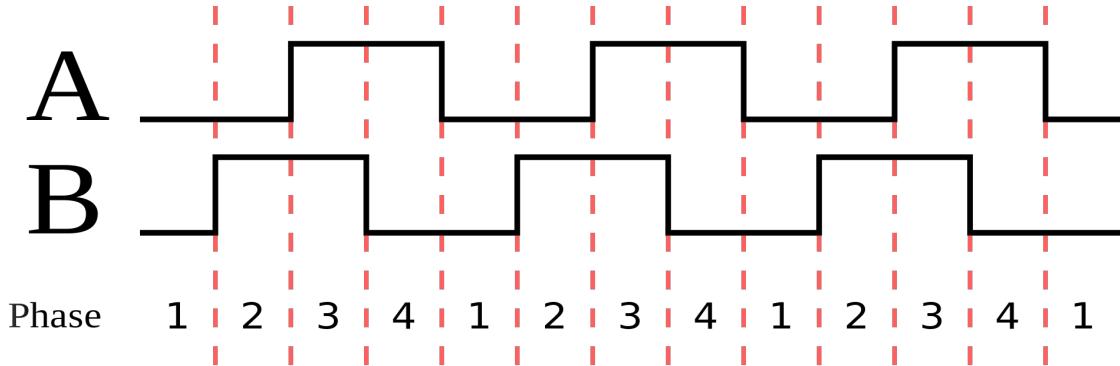


Abbildung 7: Phasenverschiebung von Pin A relativ zu Pin B

Im Programmcode würde die Implementierung folgendermaßen aussehen:

```
1 void testRotaryEncoders() {
2     ec1Val = digitalRead(RT1_ENCODER_DT);
3     //read the momentary state of Pin A
4     if (ec1Val != ec1Last) {
5         //the encoder is rotating, new state differs from old one
6         if (digitalRead(RT1_ENCODER_CLK) == ec1Val)
7             //clockwise, because A changed first
8             amin++;
9         else
10            //counter clockwise, because B changed first
11            amin--;
12        ec1Last = ec1Val;
13    }
14 }
```

Codebeispiel 4: Implementierung eines Rotary Encoders

Für das Einschalten des Weckers verwendete ich einen einfachen Kippschalter, welcher abhängig von seiner Stellung das Wecken unterbindet. Solche Schalter können jedoch nicht einfach von einer Stromquelle zum Eingangskontakt des Mikrocontrollers verbunden werden, da der Mikrocontroller, wenn die Verbindung von der Stromquelle zum Eingangskontakt durch den Schalter unterbrochen ist, kein stabiles LOW Signal liest, weil der Pin am „floaten“, zu deutsch: schweben, ist. Das bedeutet, dass kein Kontakt zur Erde oder zu einer Stromquelle gegeben ist und der Pin dadurch zufällig LOW oder HIGH liest. Um das zu verhindern, muss der Strom des Eingabepins bei Unterbrechung der Leitung über einen Widerstand zur Erde hin abfließen, damit wieder ein stabiles LOW Signal gemessen werden kann. Solch einen Widerstand nennt man Pull-Down Widerstand, da er den Strom über die Erde „herunterzieht“. Die umgekehrte Implementierung wäre der Pull-Up Widerstand, bei welchem der Schalter zur Erde verbunden ist und bei einer unterbrochenen Leitung der Eingabekontakt des Mikrocontrollers auf beispielsweise 5V gezogen wird, somit HIGH misst<sup>xvi</sup>. Glücklicherweise integriert der Arduino Uno solche Pull-Up Widerstände an seinen digitalen Kontakten, welche im Code ein- oder ausgeschaltet werden können.

---

Um den Wecker auszuschalten entschied ich mich gegen die Verwendung eines Notausschalters, da diese sehr teuer und oftmals auch sehr groß sind. Stattdessen entschied ich mich für die Verwendung einer Berührungserkennung. Ich fand dies optisch wesentlich ansprechender als einfache Taster, weshalb ich mich für diese Variante entschied. Der Materialaufwand belief sich lediglich auf einige hochohmige Widerstände, etwas Aluminiumfolie als Kontaktfläche und einiges an Zeit, um die Funktion zu testen und zu kalibrieren.

Der technische Aufbau ist verhältnismäßig simpel. Ein Kontakt des Arduino Unos wird als Ausgabepin verwendet, hinter welchem ein Widerstand im Megaohm Bereich geschaltet wird, welcher in einem Eingabepin des Arduinos endet. Zwischen Widerstand und Eingabepin kann etwas Aluminiumfolie angeschlossen werden, welche beispielsweise um das Gehäuse des Weckers gewickelt werden kann und dadurch als Sensor dient. Wenn nun der Sendepin seinen Zustand ändert (z.B. von LOW auf HIGH), misst das Programm die Zeit, welche benötigt wird, bis der Empfangspin eine Zustandsänderung von LOW auf HIGH misst<sup>XVII</sup>. Eine menschliche Hand an der Aluminiumfolie wirkt mit ihr zusammen als Kondensator. Die benötigte Zeit verhält sich proportional zu der Größe des Widerstandes und der Kapazität der menschlichen Hand. Für den Arduino Uno gibt es eine vorprogrammierte Bibliothek mit einigen Methoden, welche die Implementierung wesentlich vereinfachen, da es bei der Erkennung von Berührungen auf absolute Zuverlässigkeit ankommt und der Code der Bibliothek daher für die kürzest mögliche Laufzeit<sup>26</sup> optimiert ist. Des Weiteren bietet sie eine Möglichkeit, die Werte von Anfang an zu kalibrieren, welche in meinen Testversuchen jedoch nicht sehr zuverlässig war.

Die Methode `long capacitiveSensor(byte samples)` gibt die gemessene Zeit der oben beschriebenen Prozedur in der Anzahl an Programmdurchläufen an. Mit `byte samples` kann die Anzahl der Durchläufe bestimmt werden, um einen repräsentativeren Durchschnittswert zu bekommen. Der Wert ist vom Datentyp `long`<sup>27</sup>. Um kleinere und repräsentativere Werte zu erhalten, zieht das Programm jeweils die kürzeste gemessene Zeit von allen Werten ab, um dadurch einen relativen Zeitunterschied zu erlangen. Um diese eingebaute Kalibrierung weiter zu verbessern, implementierte ich die Methode `void touchCalibration(unsigned long sensedValue)`, welche die bereits kalibrierten Werte der Bibliothek nimmt, in ein Array der Größe zwanzig schreibt und bei einem zu großen Durchschnittswert eine Kalibrierung über die Bibliothek forciert.

---

26 Zeit, welches ein Programm bis zur einmaligen Ausführung benötigt

27 Datentyp, welcher ganze Zahlen mit bis zu 32 Bit speichert

```
1 void touchCalibration(unsigned long sensedValue) {  
2  
3     //initializing variables for touch sensor data smoothing and calibration  
4     static const int arrayLength = 20;  
5     static int index = 0;  
6     static unsigned long touchArray[arrayLength];  
7     static unsigned long touchAverage = 0;  
8     static unsigned long touchTotal = 0;  
9     static unsigned long calibrationMillis;  
10  
11    //filling the array with data and calculating average/total  
12    touchTotal = touchTotal - touchArray[index];  
13    touchArray[index] = sensedValue;  
14    touchTotal = touchTotal + touchArray[index];  
15    touchAverage = touchTotal / arrayLength;  
16    index++;  
17    if (index >= arrayLength)  
18        index = 0;  
19  
20    if (touchAverage > 3500 && millis() - calibrationMillis >= 20000) {  
21        touchControl.reset_CS_AutoCal(); // force calibration  
22        calibrationMillis = millis(); //set new time reference  
23    }  
24 }
```

#### Codebeispiel 5: Touchdaten Kalibrierung

Neben diesem Problem stieß ich jedoch auf weitere. Beispielsweise sind die gemessenen Werte extrem inkonsistent, wenn der Arduino selbst nicht geerdet ist. So gab es mit Erdung über das Aufladekabel meines Laptops einen Unterschied der kalibrierten Werte zwischen „berühren“ und „nicht berühren“ vom Faktor 200 mit Schwankungen im unberührten Zustand von  $\pm 50$ . Im nicht geerdeten Zustand schwankten die Werte jedoch mit  $\pm 1000$  bei einem Unterschied der kalibrierten Werte zwischen „berühren“ und „nicht berühren“ von Faktor 10, wobei dieser ebenfalls sehr stark schwankte. Somit war es nicht möglich, zuverlässig zwischen „berühren“ und „nicht berühren“ zu unterscheiden und eine Erdung war zwingend notwendig. Da das verwendete Neun-Volt-Netzteil jedoch keinen Erdungskontakt besaß, besorgte ich ein neues Netzteil, an welches die Kabel einfach angelötet werden mussten. Dieses besitzt anstelle von neun, fünf Volt und füttert daher direkt die Fünf-Volt-Schiene des Arduinos, indem es den eingebauten Spannungswandler umgeht. Des Weiteren besitzt es eine eingebaute Schutzschaltung, welches das alte nicht besaß.

Dadurch war der Wecker richtig geerdet, wobei in unterschiedlichen Umgebungen trotzdem unterschiedliche Werte zustande kommen, was wohl auf die unterschiedlich gute Erdung zurückzuführen ist. Durch das regelmäßige Kalibrieren ist dieses Problem jedoch um ein Vielfaches geringer.

### 3.1.5 Audiowiedergabe

Zur Wiedergabe der Töne verwendete ich den bereits angesprochenen DF-Player Mini, welcher durch seine schiere Menge an Funktionen schnell überzeugte. Zusammen mit einer zwei Gigabyte großen SD-Karte, zwei zwei Zoll großen Lautsprechern als Tieftöner an den beiden Seiten und einem kleinen Lautsprecher eines alten Smartphones als Hochtöner in der Oberseite können MP3-Dateien abgespielt werden. Das Modul kommuniziert seriell über UART, welches ich wegen einer unnötig großen und komplizierten Bibliothek händisch implementierte.

```
1 //setting up serial device
2 SoftwareSerial dfPlayer(DFPLAYER_RX, DFPLAYER_TX);
3 //filling the DF-Player array with commands
4 //Start-byte; Version; amount of bytes following; Command; Feedback;
Parameter1; Parameter2; Checksum1; Checksum2; End-byte
5 uint8_t dfCMD[10] {0x7E, 0xFF, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xEF};
6
7 //method for sending commands to the DFPlayer
8 void DFsendCommand(uint8_t cmd, uint8_t para1, uint8_t para2) {
9     dfCMD[3] = cmd;
10    dfCMD[5] = para1;
11    dfCMD[6] = para2;
12    DFchecksum();
13    dfPlayer.write(dfCMD, 10);
14    delay(50);
15 }
16
17 //method for calculating the checksum
18 void DFchecksum() {
19     uint16_t sum = 0;
20     for (int j = 1; j <= 6; j++) {
21         sum += dfCMD[j]; //adding all bytes, except start and end byte
22     }
23     sum = (0-sum); //further processing the sum, like in the original library
24     //splitting the 16-bit number into two 8-bit
25     dfCMD[7] = (uint8_t) (sum >> 8);
26     dfCMD[8] = (uint8_t) (sum);
27 }
```

Codebeispiel 6: Methoden zur UART Kommunikation mit dem DFPlayer

### 3.2 Handwerkliche Methoden

Für den Aufbau des Gehäuses baute ich zunächst ein Gerüst aus zwölf Zentimeter breitem Fichtenkantholz. Die Kanthölzer wurden sorgfältig zurecht gesägt und daraufhin miteinander verleimt, sodass trotz einiger Lücken durch Längenunterschiede der Kanthölzer von einem bis zwei

---

Millimetern, ein stabiles Gerüst von zwölf Zentimetern Höhe, sechzehn Zentimetern Breite und dreizehn Zentimetern Tiefe entstand.

Im inneren oberen Teil des Weckers verschraubte ich an den Kanthölzern eine weitere kleinere Holzplatte, auf welcher der Arduino Uno angebracht wurde, um ihn bei Bedarf herausnehmen zu können.

Für alle Seiten benutzte ich MDF-Platten mit zwei Millimetern Dicke, welche ich mit Leim und Nägeln am Gerüst befestigte. In die Frontplatte sägte ich entsprechende Löcher für die Siebensegmentanzeigen, welche möglichst eben sitzen sollten. In den kleinen Platinen auf der Rückseite der Displays ist in jeder Ecke jeweils ein kleines Loch, wodurch man eine Schraube hindurch stecken kann. Um etwas zu bieten, in das man die Schraube dann hinein schrauben kann, klebte ich für jedes Display kleine Holzkeile auf das MDF. Beim Versuch, die kleinen Schrauben daran zu befestigen, spalteten sich diese Holzkeile jedoch des Häufigeren, so dass ich die beiden kleineren Displays für das Datum und die Weckzeit schließlich doch mit etwas Kleber fixierte.

Nachdem die Position der beiden Lautsprecher bestimmt war, nahm ich für die beiden Seitenteile jeweils eine MDF-Platte, welche ich in die passenden Maße zurecht sägte und daraufhin ein rundes Loch in der Größe der Lautsprecher bohrte. Diese befestigte ich mit einem Acrylkleber. Dieser wird nach einigen Stunden extrem hart und bietet nicht nur guten Halt, sondern dichtet die Lücken auch gut ab, was für eine gute Klangqualität essentiell ist.

An der Rückseite bohrte ich lediglich ein kleines Loch für den Stromanschluss, während ich bei der Oberseite für die beiden Drehgeber und den einen Kippschalter jeweils ein Loch bohrte, bei welchen ich jedoch nicht vorsichtig genug war, was die Position dieser angeht. Daher sind die Abstände nicht ganz einheitlich.

Um die Berührungserkennung zu realisieren, klebte ich auf die MDF-Platten links, rechts und oben mit Hilfe von doppelseitigem Teppichklebeband handelsübliche Aluminiumfolie. Diese musste lediglich untereinander elektrisch verbunden sein und einen Kontakt zum Inneren, mit Hilfe eines Kabels, herstellen. Aufgrund der schnell bildenden Oxidschicht des Aluminiums, war es mir jedoch nicht möglich, ein Kabel anzulöten. Ich las, dass man das Aluminium mit einem Lötkolben punktuell erhitzen und leicht ankratzen soll, bis das Lötzinn von selbst haftet. Währenddessen dürfte das Aluminium jedoch nicht mit Sauerstoff in Kontakt kommen, was man mit Hilfe eines hitzebeständigen Öls erreichen können sollte<sup>XVIII</sup>. In meinen Versuchen kam es jedoch entweder zu einer gerissenen Aluminiumfolie oder zu unangenehm verbrannt riechendem Sonnenblumenöl, da ich kein Motor- oder Schmieröl zur Hand hatte. Daher verwendete ich schließlich einen Streifen Kupferklebeband, welches trotz seiner Klebschicht einen guten elektrischen Kontakt zu der Aluminiumfolie herstellen konnte und keine Oxidschicht ausbildet und daher lötbar ist.

Nachdem nun alle Seiten – bis auf die Unterseite – mit MDF und teilweise mit Alufolie bestückt und die Displays sowie die Lautsprecher an ihrem Platz waren, wollte ich damit beginnen, die Seiten zu furnieren bzw. zu folieren. Ich wählte Kunststofffurnier in Ahorn Optik aus dem Baumarkt, welches durch seine helle Farbe möglichst viel Licht hindurch lassen sollte. Nachdem

---

ich zwei Seiten der Box bereits foliert hatte, stellte ich jedoch fest, dass wohl ein Lösemittel meines Klebers den Kunststoff angriff. Nachdem ich diesen wechselte und erneut die Folien aufklebte, war ich mit dem Resultat jedoch in keinster Weise zufrieden. Man konnte auf den ersten Blick erkennen, dass es kein echtes Holz war, da es typische Spiegelungen gab und zudem jeder einzelne Nagel des MDFs als kleine Erhöhung sichtbar war. Ich hatte somit zwar kein Problem mit der Lichtdurchlässigkeit meines Materials, allerdings sah das Resultat optisch nicht ansprechend aus, weshalb ich die Folien schnell wieder entfernte.

Nach etwas Recherche und Gedanken über Alternativen, wie die Verwendung von milchigem Acrylglass, durch welches die Displays ebenfalls sichtbar sein würden, entschied ich mich dazu, besonders dünnes und helles Echtholzfurnier zu kaufen. Dieses ist normalerweise sehr teuer, jedoch fand ich einen Privatanbieter, welcher seine Reste für verhältnismäßig wenig Geld verkaufen wollte, was für meine Zwecke vollkommen ausreichte. Glücklicherweise war dies zudem sehr helles Ahornfurnier mit einer Dicke von etwa einem halben Millimeter.

Dieses stellte sich schließlich als ausreichend lichtdurchlässig und dadurch verwendbar heraus. Um Probleme mit Kleber zu umgehen, bestellte ich zudem einige Flaschen an besonders langsam trocknenden Cyanacrylat (CA) Kleber, besser bekannt als Sekundenkleber.

Ich furnierte immer nur eine Seite auf einmal, um auf diese besonders viel Druck ausüben zu können, damit das Furnier eng anliegt. Trotz des dickflüssigen CA-Klebers blieben mir jeweils nur etwa zwei Minuten, um das Furnier auszurichten, bis der Kleber schon anfing hart zu werden. Leider war ich beim Abschleifen des Furniers an den Kanten zu ungeduldig und unvorsichtig; teilweise verwendete ich ein Teppichmesser, welches sich jedoch als unpraktikabel herausstellte, da es die dünnen Holzfasern sehr schnell zum Reißen brachte. Das Resultat waren teilweise eingerissene Kanten, welche nicht direkt herausstechen, jedoch hätten vermieden werden können. Auffälliger sind die Risse, die nach einigen Tagen auf der Rück- und der Oberseite, wohl auf Grund von Spannungen, entstanden sind.

Schließlich behandelte ich das Furnier noch mit feinkörnigem Schleifpapier und daraufhin mit natürlicher Holzcreme auf Bienenwachs Basis, welche das sehr helle Holz leicht verdunkelt und versiegelt.

### 3.3 Elektrotechnische Methoden

Neben der rein technischen Auswahl der Hardware und deren Programmierung mussten die Einzelteile verlotet werden. Um den Aufbau weiterhin möglichst modular zu gestalten, entschied ich mich dazu, so gut es geht auf feste Lötstellen zu verzichten und stattdessen Steckkabel zu verwenden. Für die Anbindung des DFPlayers und der Real Time Clock war die Verwendung von Kabeln jedoch nicht denkbar, weshalb ich mich dafür an dem Design der Arduino Shields orientierte. Dies sind kleine Platinen mit extra Funktionalität, welche auf einen bestehenden Arduino gesteckt werden können und über die Pins an der Seite Kontakt herstellen.

Für die Umsetzung wählte ich eine doppelseitige Lochrasterplatine, durch welche ich die beiden Module hindurch stecken und festlöten konnte. Durch das richtige Legen der Kontaktpfade ist die Platine schließlich durch einfaches Aufstecken verwendbar, indem sowohl die Stromversorgung, als auch die Kommunikation alleine über die Steckverbindungen ablaufen.

Um die Displays, die Lautsprecher und die Bedieneinheiten zu verbinden, lötete ich an die Kontakte jeweils ein Kabel mit Steckverbindung, welche in den Eingangskontakten des Arduinos enden. Die Stromversorgung stellte ich global über fest verlötete Kabel sicher, welche zu jedem Verbraucher einzeln führen.

## 4 Ergebnisse

Für das Resultat entwickelte ich einen Schaltplan, mit dessen Hilfe das Projekt elektrotechnisch einfach nachzuvollziehen und nachzubauen ist. Des Weiteren erstellte ich Zeichnungen, ähnlich einer Bedienungsanleitung, an welchen man alle beschriebenen Elemente und deren Position erkennen kann. Diese befinden sich im Maßstab eins zu zwei. Zu guter Letzt gibt es noch tatsächliche Fotos des fertigen Produktes und den gesamten Quellcode in acht Teilen.

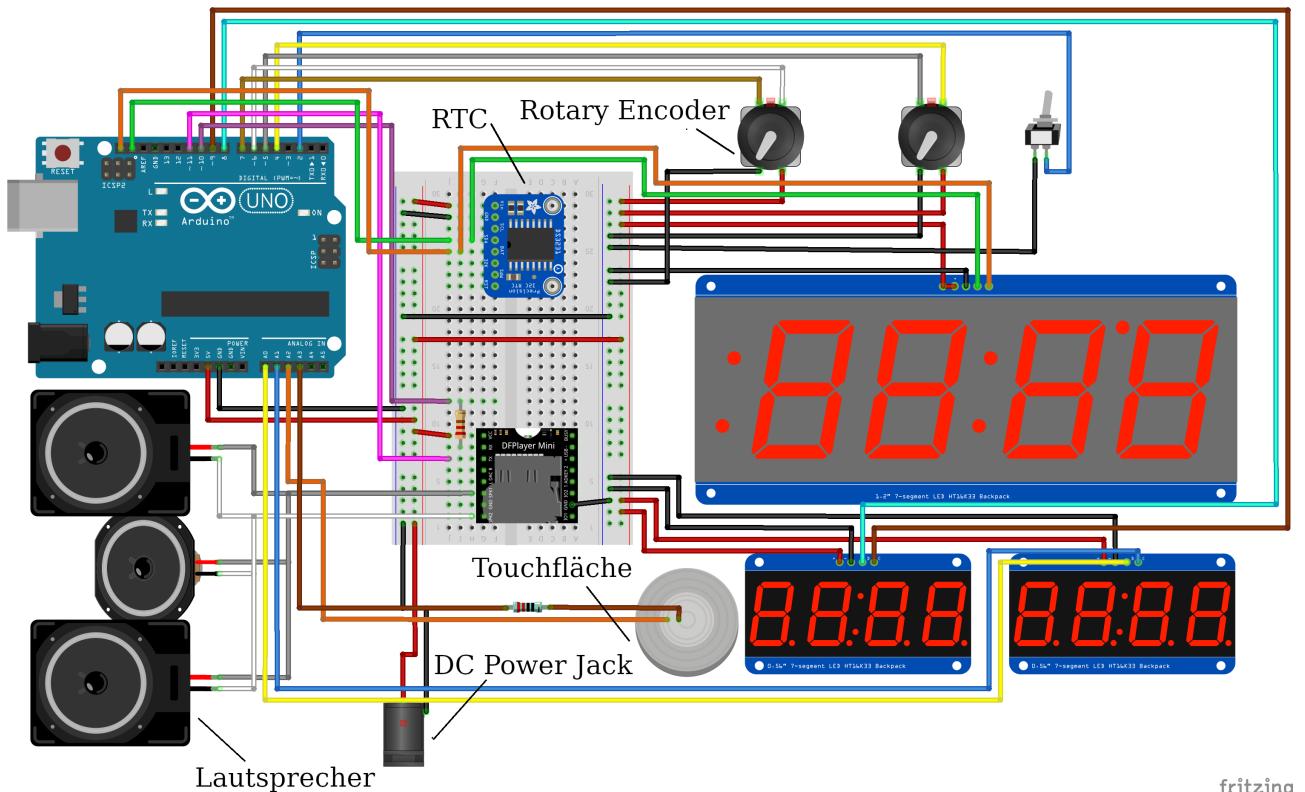


Abbildung 8: Schematische Darstellung der Verkabelung. Gleiche Farben entsprechen nicht zwingend gleichen Leitungen

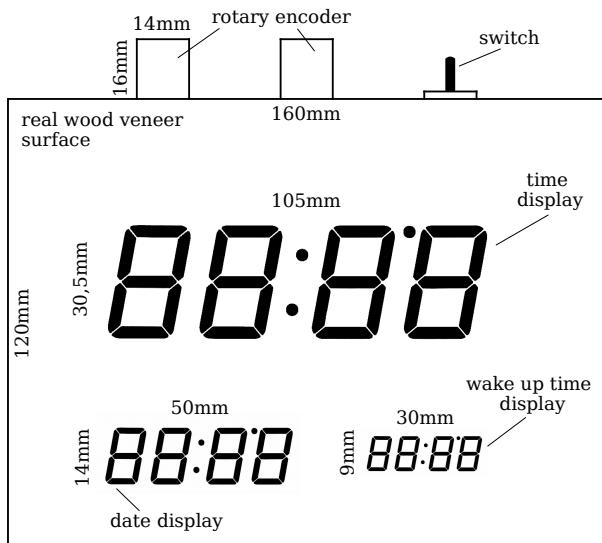


Abbildung 9: Frontansicht

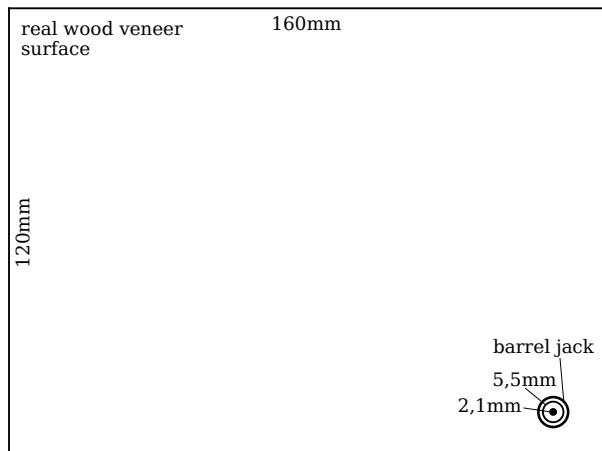


Abbildung 10: Rückansicht

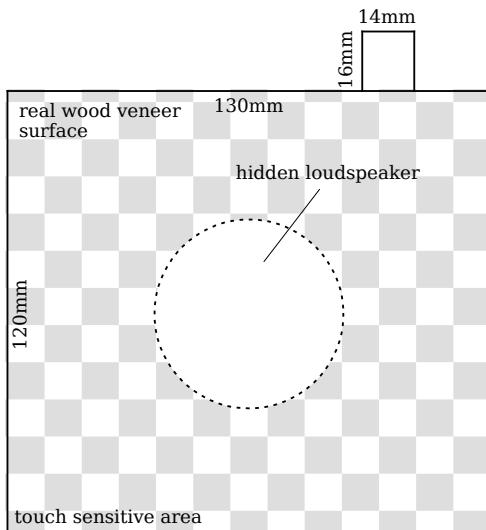


Abbildung 11: Seitenansicht, links

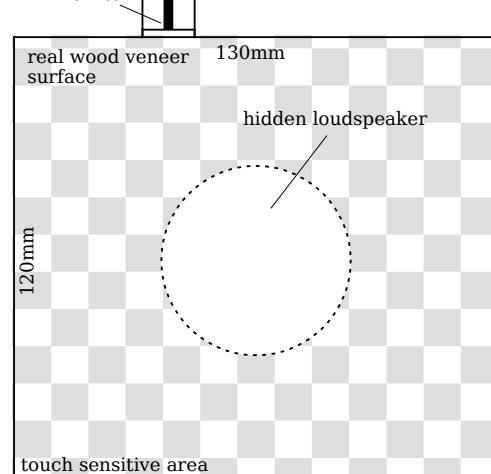


Abbildung 12: Seitenansicht, rechts

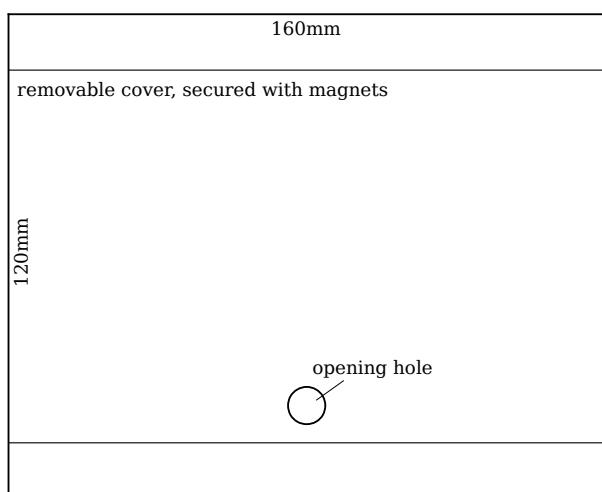


Abbildung 13: Ansicht von unten

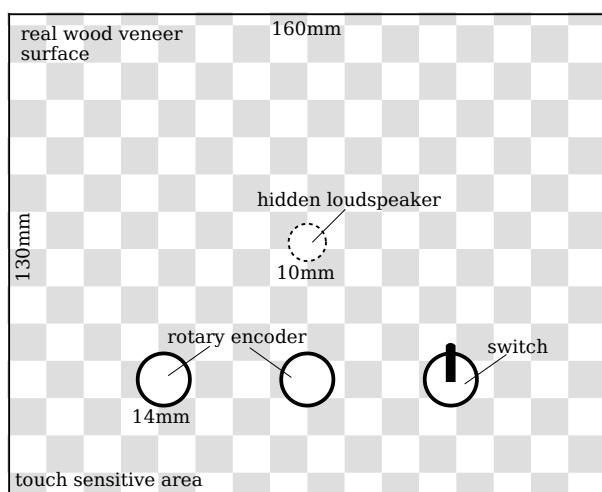


Abbildung 14: Draufsicht



Abbildung 15: Frontalansicht, eingeschaltet



Abbildung 16: Frontalansicht, ausgeschaltet



Abbildung 17: Schrägansicht, eingeschaltet



Abbildung 18: Schrägansicht, ausgeschaltet



Abbildung 19: Schrägansicht, eingeschaltet



Abbildung 20: Schrägansicht, ausgeschaltet

```
1 #include <TimeLib.h>
2 #include <DS3232RTC.h>
3 #include <Wire.h>
4 #include <Timezone.h>
5 #include <TM1637Display.h>
6 #include <Adafruit_LEDBackpack.h>
7 #include <SoftwareSerial.h>
8 #include <CapacitiveSensor.h>
9
10 CapacitiveSensor touchControl = CapacitiveSensor(17,16);
11 //Daylight saving time via "Timezone.h"
12 //creating the statements
13 TimeChangeRule myDST = {"MESZ", Last, Sun, Mar, 2, +120};
14 TimeChangeRule mySDT = {"MEZ", Last, Sun, Oct, 3, +60};
15 //initializing object DE of class "Timezone" with the attributes from above
16 Timezone DE(myDST, mySDT);
17 //giving the pointer to tcr to TimeChangeRule
18 TimeChangeRule *tcr;
19 //initializing utc and local time of type time_t
20 time_t utc, local;
21
22 //defining the pin for the alarm flip switch
23 #define FLIP_SWITCH 2
24 //defining the pins for the alarm display
25 #define ALARM_DISPLAY_CLK 14
26 #define ALARM_DISPLAY_DIO 15
27 //defining the pins for the date display
28 #define DATE_DISPLAY_CLK 9
29 #define DATE_DISPLAY_DIO 8
30 //defining the pins for the rotary encoders 1 and 2
31 #define RT1_ENCODER_CLK 6
32 #define RT1_ENCODER_DT 7
33 #define RT2_ENCODER_CLK 4
34 #define RT2_ENCODER_DT 5
35 //defining the pins for the DF Player
36 #define DFPLAYER_RX 10
37 #define DFPLAYER_TX 11
38
39 #define SONG_LIMIT 400
40
41 //initializing variables for getting the rotary encoders position
42 int ec1Last;
43 int ec1Val;
44 int ec2Last;
45 int ec2Val;
46 //initializing variable for the alarm hour and alarm minute
47 int ahour;
48 int amin;
49 //initializing variable for time reference
50 unsigned long encoderMillis;
51 //initializing boolean for alarm
52 boolean playing = false;
```

Codebeispiel 7: Sourcecode (Teil 1)

```
53 boolean displayON = true;
54
55 //filling the DF-Player array with commands
56 //Start-byte; Version; amount of bytes following; Command; Feedback;
Parameter1; Parameter2; Checksum1; Checksum2; End-byte
57 uint8_t dfCMD[10] {0x7E, 0xFF, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00,
0xEF};
58
59 //initialize dfPlayer
60 SoftwareSerial dfPlayer(DFPLAYER_RX, DFPLAYER_TX);
61 //initialize the alarm display
62 TM1637Display alarmDisplay(ALARM_DISPLAY_CLK, ALARM_DISPLAY_DIO);
63 //initialize the date display
64 TM1637Display dateDisplay(DATE_DISPLAY_CLK, DATE_DISPLAY_DIO);
65 //Initializing a variable for the clock display
66 Adafruit_7segment clockDisplay = Adafruit_7segment();
67
68 void setup() {
69     //begin serial communictaion with PC (Baudrate 115200Hz)
70     Serial.begin(115200);
71     Serial.setTimeout(0);
72
73     //setting up an unused analog pin for random seeding for random playback
74     randomSeed(analogRead(5));
75     //begin serial communictaion with the DFPlayer
76     dfPlayer.begin(9600);
77     if (dfPlayer.isListening() == true) {
78         Serial.println("DFPlayer is online!");
79     }
80
81     //print indicator for touch-sensed values
82     Serial.println("Touch-sensed values:");
83     Serial.print("raw");
84     Serial.print("\t");
85     Serial.print("auto");
86     Serial.print("\t");
87     Serial.print("cycle");
88     Serial.print("\t");
89     Serial.print("sum");
90     Serial.print("\t");
91     Serial.print("avg");
92     Serial.println();
93
94     //begin serial communictaion with the clock display, Hardware ID 0x70
95     clockDisplay.begin(0x70);
96
97     //setting the brightness
98     alarmDisplay.setBrightness(3, true); //up to 7
99     dateDisplay.setBrightness(3, true); //up to 7
100    clockDisplay.setBrightness(16); //up to 16
101
102    //setting the pin configuration of the flip switch with Pullup resistor
```

Codebeispiel 8: Sourcecode (Teil 2)

```
103  pinMode(CLIP_SWITCH, INPUT_PULLUP);
104
105 //setting the pin configurations of the rotary encoders
106 pinMode(RT1_ENCODER_CLK, INPUT);
107 pinMode(RT1_ENCODER_DT, INPUT);
108 pinMode(RT2_ENCODER_CLK, INPUT);
109 pinMode(RT2_ENCODER_DT, INPUT);
110 ec1Last = digitalRead(RT1_ENCODER_DT);
111 ec2Last = digitalRead(RT1_ENCODER_DT);
112
113 //Sync the internal clock
114 setSyncProvider(RTC.get());
115 setSyncInterval(3600); //sync every hour
116 //print the alarm and date display
117 displayPrintAlarm();
118 displayPrintDate();
119
120 DFsendCommand(0x09, 0x00, 0x02); //playback source TF-Card
121 DFsendCommand(0x0F, 0x00, 0x01); //select folder 1
122
123 touchControl.set_CS_Autocal_Millis(10000); //autocalibration every 10s
124 touchControl.set_CS_Timeout_Millis(4000); //wait max 4s
125 }
126
127 void loop() {
128 //setting the RTC time from the serial port
129 setRTCTime();
130 //sensedValueing, if the encoders turned
131 testRotaryEncoders();
132 //reference for calling the subroutine below every 5s
133 static unsigned long syncMillis;
134 if ((millis() - syncMillis) >= 5000) {
135   utc = now(); //set utc time
136   local = DE.toLocal(utc, &tcr); //convert utc to local
137   //print time
138   clockDisplayPrint();
139   static unsigned long alarmStartMillis; //refrence to start of alarm
140   if (playing == false) {
141     alarmStartMillis = testForAlarm();
142   }
143   //increase volume by one every 0,2s after start of alarm
144   static int j = 1;
145   if (playing == true) {
146     if (j < 30) {
147       if (millis() - alarmStartMillis >= (j * 200)) {
148         DFsendCommand(0x04, 0x00, 0x00); //increase volume by one
149         j++;
150       }
151     }
152   } else {
153     playing = false; //set playing to false after timeout

```

Codebeispiel 9: Sourcecode (Teil 3)

```
154     }
155   }
156   else {
157     j = 1; //set j back to 1
158   }
159
160   syncMillis = millis(); //time reference for calling this subroutine
161   displayPrintDate(); //print date
162 }
163 //test for touch every 125ms if the encoders were not in use for 2,5s
164 static unsigned long touchMillis;
165 if (millis() - touchMillis >= 125 && millis() - encoderMillis >= 2500){
166   testForTouching(); //test if device got touched
167   touchMillis = millis(); //time reference
168 }
169 }
170
171 void testForTouching() {
172   //get touch sensor value
173   unsigned long sensedValue = touchControl.capacitiveSensor(100);
174
175   //stop playback if being touched
176   if (sensedValue > 5000 && playing == true) {
177     DfSendCommand(0x0E, 0x00, 0x00); //stop Playback
178     playing = false;
179   }
180
181   touchCalibration(sensedValue); //calibrate touch values
182
183   //higher the brightness at night when touched
184   if (hour(local) >= 21 || hour(local) <= 10) {
185     if (sensedValue >= 4000 && playing == false && displayON == false) {
186       alarmDisplay.setBrightness(2, true);
187       dateDisplay.setBrightness(2, true);
188       clockDisplay.setBrightness(12);
189       displayON = true;
190       encoderMillis = millis();
191     }
192     //lower the brightness at night
193     else if (displayON == true && playing == false) {
194       alarmDisplay.setBrightness(2, false);
195       dateDisplay.setBrightness(2, false);
196       clockDisplay.setBrightness(0);
197       displayON = false;
198     }
199   }
200   //higher the brightness at day
201   else if (displayON == false) {
202     alarmDisplay.setBrightness(3, true); //up to 7
203     dateDisplay.setBrightness(3, true); //up to 7
204     clockDisplay.setBrightness(16); //up to 16
205     displayON = true;
```

Codebeispiel 10: Sourcecode (Teil 4)

```
206     }
207
208     //print time, date and alarm
209     clockDisplayPrint();
210     displayPrintDate();
211     displayPrintAlarm();
212 }
213
214 //method for calibrating touch values
215 void touchCalibration(unsigned long sensedValue) {
216
217     //initializing variables for touch sensor data smoothing and calibration
218     static const int arrayLength = 20;
219     static int index = 0;
220     static unsigned long touchArray[arrayLength];
221     static unsigned long touchAverage = 0;
222     static unsigned long touchTotal = 0;
223
224     touchTotal = touchTotal - touchArray[index];
225     touchArray[index] = sensedValue;
226     touchTotal = touchTotal + touchArray[index];
227     touchAverage = touchTotal / arrayLength;
228
229     //printing different values for debugging
230     Serial.print(touchControl.capacitiveSensorRaw(100));
231     Serial.print("\t");
232     Serial.print(sensedValue);
233     Serial.print("\t");
234     Serial.print(index);
235     Serial.print("\t");
236     Serial.print(touchTotal);
237     Serial.print("\t");
238     Serial.println(touchAverage);
239
240
241     index++; //increase array index by one
242     if (index >= arrayLength)
243         index = 0; //reset to zero
244     //recalibrate if average value is too high, max every 20s
245     static unsigned long calibrationMillis;
246     if (touchAverage > 3500 && millis() - calibrationMillis >= 20000) {
247         touchControl.reset_CS_AutoCal(); // calibrate via library
248         calibrationMillis = millis();
249     }
250 }
251
252 //testing for rotary encoder movement
253 void testRotaryEncoders() {
254     ec1Val = digitalRead(RT1_ENCODER_DT);
255     if (ec1Val != ec1Last) {
256         //the encoder is rotating
```

Codebeispiel 11: Sourcecode (Teil 5)

```
257     if (digitalRead(RT1_ENCODER_CLK) == ec1Val) //clockwise
258         amin++;
259     else
260         amin--;
261     ec1Last = ec1Val;
262     encoderMillis = millis();
263     //print alarm time and set high brightness
264     alarmDisplay.setBrightness(3, true);
265     clockDisplay.setBrightness(16);
266     displayON = true;
267     displayPrintAlarm();
268 }
269
270 ec2Val = digitalRead(RT2_ENCODER_DT);
271 // TODO: sensedValue while loop, instead of if
272 if (ec2Val != ec2Last) {
273     //the encoder is rotating
274     static short i = 0;
275
276     if (digitalRead(RT2_ENCODER_CLK) == ec2Val) //clockwise
277         i++;
278     else //counter clockwise, because B changed first
279         i--;
280     ec2Last = ec2Val;
281     encoderMillis = millis();
282
283     //increasing hour only every two steps
284     if (i%2 == 0) {
285         ahour = ahour + (i/2);
286         i = 0;
287     }
288     //print alarm time and set high brightness
289     alarmDisplay.setBrightness(3, true);
290     clockDisplay.setBrightness(16);
291     displayON = true;
292     displayPrintAlarm();
293 }
294 }
295
296 //display the time
297 void clockDisplayPrint() {
298     //make one variable out of hours and minutes
299     int timeValue = hour(local)*100 + minute(local);
300     //display the given time with dots
301     clockDisplay.print(timeValue, DEC);
302     //Serial.println(timeValue);
303     //print zeros when given hour is <10 || <1
304     if (timeValue < 1000)
305         clockDisplay.writeDigitNum(0, 0);
306     if (timeValue < 100)
307         clockDisplay.writeDigitNum(1, 0);
```

Codebeispiel 12: Sourcecode (Teil 6)

```
308 if (timeValue < 10)
309     clockDisplay.writeDigitNum(3, 0);
310     clockDisplay.drawColon(true); //print with colon
311     clockDisplay.writeDisplay(); //write to I2C Bus
312 }
313
314 // code to process time sync messages from the serial port
315 // on linux, type: stty -F /dev/ttyACM0 cs8 115200 ignbrk -brkint -imaxbel
316 // -opost -onlcr -isig -icanon -iexten -echo -echoe -echok -echoctl -
317 // echoke noflsh -ixon -crtscs && date +T%> /dev/ttyACM0
318 #define TIME_HEADER 'T' // Header tag from the message
319 //setting the RTC from the serial port
320 void setRTCTime() {
321     if (Serial.available()) { //test if serial communication is established
322         if (Serial.find(TIME_HEADER)) { //look for the "T"
323             unsigned long pcTime = Serial.parseInt(); //parse the integer
324             RTC.set(pcTime); //set RTC with UNIX time
325             setTime(pcTime); //set local time with UNIX time
326             Serial.println(pcTime);
327         }
328     }
329 }
330
331 //prevent hour and minute variable from having impossible values
332 void alarmConversion() {
333     if (ahour >= 24)
334         ahour = 0;
335     if (ahour <= -1)
336         ahour = 23;
337
338     if (amin >= 60) {
339         amin = 0;
340         ahour++;
341     }
342     if (amin <= -1) {
343         amin = 59;
344         ahour--;
345     }
346 }
347
348 //check if the alarm should ring
349 unsigned long testForAlarm() {
350     if (ahour == hour(local) && amin == minute(local) && (millis() - encoderMillis) >= 5000 && digitalRead(FLIP_SWITCH) == LOW) {
351         Serial.println("ALARM!");
352         DFsendCommand(0x03, 0x00, random(0, SONG_LIMIT)); //random song
353         DFsendCommand(0x01, 0x00, 0x00); //play song
354         DFsendCommand(0x06, 0x00, 0x01); //specify volume to level 1
355
356         playing = true;
357         alarmDisplay.setBrightness(3, true); //up to 7
358         dateDisplay.setBrightness(3, true); //up to 7
```

Codebeispiel 13: Sourcecode (Teil 7)

```
359     clockDisplay.setBrightness(16); //up to 16
360     displayON = true;
361
362     //print time, date and alarm
363     clockDisplayPrint();
364     displayPrintDate();
365     displayPrintAlarm();
366
367     return millis(); //return for time reference
368 }
369 }
370
371 void displayPrintDate() {
372     int date = day(local) * 100 + month(local); //make one variable out of
373     day and month
374     dateDisplay.showNumberDecEx(date, 0x40, true, 4, 0); //print with dot
375 }
376 //display the alarm time
377 void displayPrintAlarm() {
378     alarmConversion();
379     alarmDisplay.showNumberDecEx(ahour, 0x10, true, 2, 0); //print alarm
380     hours with dot
381     alarmDisplay.showNumberDecEx(amin, 0x10, true, 2, 2); //print alarm
382     minutes with dot
383 }
384
385 //method for calculating the checksum
386 void DFchecksum() {
387     uint16_t sum = 0;
388     for (int j = 1; j <= 6; j++) {
389         sum += dfCMD[j]; //adding all bytes, except start and end byte
390     }
391     sum = (0-sum); //further processing the sum, like the original library
392
393 }
394
395 //method for sending commands to the DFPlayer
396 void DFsendCommand(uint8_t cmd, uint8_t para1, uint8_t para2) {
397     dfCMD[3] = cmd;
398     dfCMD[5] = para1;
399     dfCMD[6] = para2;
400     DFchecksum();
401     dfPlayer.write(dfCMD, 10); //write on serial bus
402     delay(50); //delay for 50ms
403 }
```

Codebeispiel 14: Sourcecode (Teil 8)

## 5 Diskussion

### 5.1 Reflektion der Herangehensweise

Ich begann die Entwicklung meines Produktes auf einer theoretischen Ebene, entwickelte ein Konzept und schloss davon ausgehend auf benötigte Teile und wägte Machbarkeit und Kosten mit Nutzen ab.

Nach der Vervollständigung meines Konzepts überlegte ich mir für die technische Umsetzung ein Schaltdiagramm der Einzelteile und begann daraufhin dieses auf Steckplatten zu testen, während ich den Programmcode zu schreiben begann. Ich entschied mich für diese Art der Herangehensweise, um gegebenenfalls Änderungen am Konzept oder den Einzelteilen vornehmen zu können, ohne ein neues Gehäuse oder eine neue Platine bauen bzw. löten zu müssen.

Nachdem ich mir sicher war, an den wichtigen Bauteilen nichts mehr ändern zu wollen begann ich damit, die Platine zu löten.

Mit Vervollständigung der eben genannten Schritte begann ich schließlich das tatsächliche Gehäuse zu bauen, bei welchem ich während der Entwicklung immer wieder die Bauteile einsetzte um es mit dem Konzept abzulegen bzw. die Passform zu überprüfen.

Schließlich verbaute ich die Displays zusammen mit dem Arduino und den Lautsprechern im Gehäuse und lötete Kabel an diese, um sie zu verbinden.

Diese Herangehensweise, welche von einem theoretischen Konzept ausgeht, daraufhin vorsieht das Technische umzusetzen, den Programmcode zu schreiben und schließlich handwerklich das Gehäuse zu bauen, bewerte ich rückblickend als sehr positiv und passend. In der Softwareentwicklung wird sie mit dem Wasserfallmodell beschrieben, wobei dieses eben auch auf Projekte wie in meinem Fall zu übertragen ist. Es sieht eine festgelegte Reihenfolge der einzelnen Schritte bis hin zum Ziel, mit jeweils klar definierten Anforderungen und daraus folgenden Ergebnissen vor<sup>XIX</sup>. Dadurch war in meinem Fall sichergestellt, dass bei einer Änderung in Mitten des Prozesses der Produktentwicklung diese auch ohne besonders viel Aufwand durchgeführt werden konnte. Lediglich beim Handwerklichen fiel mir, während ich mein Konzept des Holzfurniers überdachte, auf, wie dieses Problem der Umsetzung unter Umständen viel Arbeit machen könnte, wenn ich mich dazu entschieden hätte, die Darstellung entsprechend der Umstände zu verändern. Da ich jedoch schließlich die Möglichkeit fand, den Wecker in Echtholzfurnier zu kleiden, musste ich keine Änderungen am Konzept vornehmen, weshalb mit dieser Arbeit erspart blieb.

Da ich Probleme beim Prozess der handwerklichen Umsetzung jedoch am ehesten vorhersehen kann, bzw. diesen am besten einschätzen konnte, sehe ich auch rückblickend die oben genannte Reihenfolge als sinnvoll an und würde sie bei einem ähnlichen Projekt übernehmen.

## 5.2 Reflektion von gewonnenen Erkenntnissen

Neben dem tatsächlichen Bau des Weckers habe ich auch einen beachtlichen Erkenntnisgewinn zu verzeichnen.

Bei der Integrierung und Installation der benötigten Einzelteile lernte ich viel über die fundamentalen Funktionsweisen mikroelektronischer Bauteile, wie die Art der Kommunikation über serielle Schnittstellen, da ich eigenständig eines dieser Protokolle implementierte.

Im Bereich der Programmierung lernte ich, die gewonnenen Erkenntnisse über die Funktionsweisen in Code umzusetzen und meine eigenen Vorstellungen eines Weckers einzubringen. Da meine Vorkenntnisse in diesem Bereich lediglich durch den Schulunterricht geprägt waren, welcher sich zudem nur auf die Verwendung der hohen Programmiersprache Java<sup>28</sup> beschränkte, ließ ich mir grundlegende Konzepte maschinennaher Programmiersprachen, wie z.B. Pointer<sup>29</sup> erklären und lernte, meinen Code in seiner Laufzeit zu optimieren, sodass die begrenzten Ressourcen ausreichten. Des Weiteren lernte ich über die Implementierung von Prüfsummen<sup>30</sup> beim analysieren und reimplementieren des Quellcodes der DFPlayer Bibliothek. Internetforen, Wikieinträge, der direkte Kontakt zu Softwareentwicklern und Bücher zum Thema halfen mir, mein Programm zu schreiben und in Situationen, in denen ich nicht weiter kam, eine Lösung zu finden. Zudem war dies mein erstes großes Programm, wodurch ich lernte, meine Vorstellungen zu strukturieren und Stück für Stück umzusetzen.

Damit die kleinen Einzelteile miteinander kommunizieren können und mit Strom versorgt werden, musste ich sie zum Großteil verlöten. Da die Kontakte sehr klein und die Struktur doch schon relativ komplex auf meiner eigens entwickelten Platine waren, erforderte es eine sehr ruhige Hand sowie einige Anläufe, um durchgehenden und stabilen Kontakt herzustellen.

Im Handwerklichen lernte ich erneut über die Bedeutsamkeit der Genauigkeit, da schon kleine Abweichungen beim Sägen im Bereich von einem halben Millimeter zu großen Unterschieden im Gesamtbild führten, welche ich im Nachhinein mit wesentlich mehr Arbeit korrigieren musste. Des Weiteren lernte ich wie Furnier zu verarbeiten und zu handhaben ist, sowie über die Verwendung eines geeigneten Klebers.

Des Weiteren lernte ich den Prozess der Produktentwicklung ein wenig kennen. Ich startete mit einer einfachen Idee, fing an, diese gemäß des Wasserfallmodells in ein Konzept, daraufhin in einen Prototypen und schlussendlich in ein fertiges Produkt zu überführen. Dies war mir wegen des Business@School<sup>31</sup> Projektes schon ein wenig vertraut, jedoch besaß der hier gebaute Wecker ein um ein Vielfaches größeres Ausmaß, als das damals in der Gruppe entwickelte Produkt. So managte

28 Seit 1995 von Oracle entwickelte objektorientierte Programmiersprache, welche mit Hilfe von Virtualisierung rekomplizieren obsolet macht und dadurch plattformunabhängig ist

29 Zu deutsch: Zeiger; zeigt mit seinem Wert auf die Speicheradresse einer Variable (oder anderen Entität) und ermöglicht damit beispielsweise die Implementierung dynamisch zugewiesenen Speichers

30 Ein Wert, welcher aus Ausgangsdaten berechnet wird und mit dessen Hilfe die Integrität dieser Daten sichergestellt werden kann

31 Eine Initiative der Boston Consulting Group, Schüler mit Startups und Geschäftsideen vertraut zu machen

ich von Anfang bis Ende eigenständig den Prozess der Entwicklung, wobei ich auf Dinge wie beispielsweise mein begrenztes Budget oder geringe Programmiererfahrung achten musste. Dadurch lernte ich, wie beim Kauf von Hardwarekomponenten zu sparen ist und wie ich mich im Bereich der Programmierung weiterbilden kann.

Insgesamt steckte extrem viel Arbeit in meinem Projekt. Ich saß etwa ein dreiviertel Jahr an der Entwicklung des Weckers, wobei es zwischendurch Phasen mit mehr und mit weniger Aktivität gab. Am meisten Zeit beanspruchte die Entwicklung der Software. Zwar beschränkt sie sich das Programm lediglich auf vierhundert Zeilen Code, jedoch beinhaltet dieser eine Vielzahl an unterschiedlichen Methoden und die Entwicklung involvierte die Auseinandersetzung mit zuvor beschriebenen Konzepten.

Grob überschlagen würde ich die Arbeitszeit für das Schreiben der Software und das Debuggen dieser auf ein halbes Jahr, bei einer Stunde Arbeit pro Tag schätzen. Dies entspräche dann etwas weniger als zweihundert Stunden. Darin eingeslossen ist logischerweise auch Recherche und Weiterbildung in diesem Bereich, da dies immer abwechselnd geschah und miteinander verknüpft ist. Zudem machte mir die Implementierung und Kalibrierung der Berührungserkennung gegen Ende viel Arbeit, da sich dies schwieriger als gedacht erwies.

Die benötigte Zeit für die Recherche der benötigten Teile und das Einkaufen dieser würde ich lediglich auf etwa fünf bis zehn Stunden schätzen, da sich dies doch als verhältnismäßig einfach gestaltete, nachdem ich ein Konzept vorliegen hatte.

Was hingegen wieder viel Zeit in Anspruch nahm war das physische Bauen des Weckers. Hierbei hätte der Zugriff auf einen Lasercutter oder ähnliches massiv geholfen. Durch händisches Sägen, genaues Arbeiten, langsam trocknenden Holzleim und nachträgliches händisches Ausbessern benötigte ich mehr Zeit, als eigentlich nötig gewesen wäre – etwa zwanzig bis dreißig Stunden.

Zusammengefasst schätze ich die benötigte Zeit somit auf etwa zweihundert bis zweihundertdreißig Stunden, wobei diese Primär dem Erkenntnisgewinn diente und bei einem ähnlichen Projekt vermutlich um einiges geringer ausfallen würde.

## 5.3 Bewertung des Produktes

Wenn ich den Wecker mit meinen ursprünglichen Anforderungen vergleiche, wurden diese alle erfüllt. Jedoch gibt es teilweise Verbesserungsmöglichkeiten bei der Umsetzung der einzelnen Funktionen.

Was ich für ein nächstes Mal implementieren würde, wäre der zuvor benannte Funkempfänger für die Synchronisation der Uhrzeit. Die verwendete Real Time Clock verrichtet zwar zuverlässig ihre Arbeit, jedoch muss man sie nach zwei bis drei Jahren neu stellen, da nach dieser Zeit die Uhrzeit um etwa eine Minute abweicht, was mindestens für ein kommerzielles Produkt nicht meiner

---

gewünschten Benutzerfreundlichkeit entspricht. Durch die Verwendung des DCF77 Signals wäre dieser Schritt redundant.

Die von mir verwendeten Lautsprecher nahm ich aus meinem Bestand, welche jedoch dadurch auch eine minderwertige Qualität besitzen. Zwar ist die Klangqualität besser als bei generischen Radioweckern oder ähnlichen Produkten des niedrigen Preissektors, jedoch besitzt mein Produkt einen höheren Preis, welcher durch qualitativ hochwertigere Lautsprecher eher gerechtfertigt wäre und dem Gesamtbild besser entsprechen würde.

Die äußere Erscheinung und die Verwendung des Holzfurniers empfinde ich als sehr gelungen umgesetzt. Die Displays scheinen sehr deutlich und gut lesbar durch das Furnier hindurch und durch die Abdunklung bei Nacht stören diese auch nicht beim Einschlafen. Des Weiteren ist die Haptik durch die Verwendung des echten Holzes dem entsprechend und daher um einiges hochwertiger als eine künstliche Alternative in Form von Acryl oder Vinyl, welche in vergleichbaren kommerziellen Produkten oftmals verwendet werden, wie in meinem Vergleich zu Beginn beschrieben. Einzig zu bemängeln wären die kleinen Risse, welche an der Ober- und Rückseite im Furnier entstanden sind. Vermutlich ausgelöst durch Spannungen im Holz zusammen mit dem schnell und hart trocknenden CA-Kleber. Darauf würde ich bei einem weiteren Mal besser achten.

Des Weiteren bewerte ich die Implementierung der Touchfunktion unter dem Furnier als optisch sehr gelungene Lösung. Leider ist jedoch die Zuverlässigkeit dieser stark abhängig von dem vorhandenen Erdungspotential und der Qualität dieses. So soll dieses gerade in städtischen Umgebungen oftmals durch nahegelegene Bahntrassen stark gestört werden<sup>xx</sup>. Um die Zuverlässigkeit in diesem Bereich weiter zu verbessern, müsste ich weitergehende Recherchen betreiben und weitere errungene Kenntnisse anwenden.

Zusammengerechnet kosteten die Einzelteile etwa einhundert Euro, was sich noch im von mir gesetzten finanziellen Rahmen befindet.

## 6 Materialliste

Für das vollendete Produkt wurden folgende Materialien in zugehörigen Mengen verwendet:

- 3x MDF-Platte 200mm x 400mm x 2mm
- 2x Fichtenkantholz 1000mm x 12mm x 12mm
- 4x Ahornfurnier 210mm x 297mm x 0,5mm
- 1x Rotes Filz 120mm x 160mm
- 1x Aluminiumfolie 130mm x 400mm
- etwas Kupferklebeband

- 2x CA-Kleber 20g
- etwas doppelseitiges Teppichklebeband
- etwas Holzleim
- einige Nägel und Schrauben
- 1x Arduino Uno R3
- 1x DFPlayer Mini
- 1x 2GB Mikro SD-Karte
- 2x Tiefen Lautsprecher 2“
- 1x Höhen Lautsprecher
- 1x DS3231 Real Time Clock
- 1x Wiederaufladbare LIR2032 Zelle
- 2x Rotary Encoder KY-040
- 1x Kippschalter
- 1x Adafruit 1,2“ vierziffrige Siebensegmentanzeige
- 1x TM1637 0,56“ vierziffrige Siebensegmentanzeige
- 1x TM1637 0,36“ vierziffrige Siebensegmentanzeige
- 1x DC Power Jack Male 2,1mm
- 1x Meanwell 5V 2,6A Netzteil
- 1x Schutzkontaktstromkabel
- 1x DC Power Jack Female 2,1mm
- 1x Doppelseitige Lochrasterplatine
- einige Zentimeter Kupferdraht
- einige Steckkabel
- etwas Schrumpfschlauch und Isolierklebeband
- 1x USB B Kabel
- 3x Magnet
- etwas Schleifpapier in Körnungen von 80-400
- etwas Bienenwachs Holzversiegelung
- etwas Lötzinn

## 7 Verzeichnisse

### Tabellenverzeichnis

Tabelle 1: Gegenüberstellung verschiedener Wecker.....	5
Tabelle 2: Gegenüberstellung verschiedener Entwicklungsplattformen.....	7

### Codeverzeichnis

Codebeispiel 1: Einmaliges Einrichten via stty des Arduino Terminals, eingehängt an /dev/ttyACM0 mit einer Abtastrate von 115200 Hz.....	13
Codebeispiel 2: Weiterleitung der Ausgabe von date an das Terminal des Arduinos (/dev/ttyACM0) .....	13
Codebeispiel 3: Implementierung der Zeitumstellung.....	13
Codebeispiel 4: Implementierung eines Rotary Encoders.....	15
Codebeispiel 5: Touchdaten Kalibrierung.....	17
Codebeispiel 6: Methoden zur UART Kommunikation mit dem DFPlayer.....	18
Codebeispiel 7: Sourcecode (Teil 1).....	24
Codebeispiel 8: Sourcecode (Teil 2).....	25
Codebeispiel 9: Sourcecode (Teil 3).....	26
Codebeispiel 10: Sourcecode (Teil 4).....	27
Codebeispiel 11: Sourcecode (Teil 5).....	28
Codebeispiel 12: Sourcecode (Teil 6).....	29
Codebeispiel 13: Sourcecode (Teil 7).....	30
Codebeispiel 14: Sourcecode (Teil 8).....	31

### Abbildungsverzeichnis

Abbildung 1: Produktbild.....	5
Abbildung 2: Produktbild.....	5
Abbildung 3: Produktbild.....	5
Abbildung 4: Produktbild.....	7
Abbildung 5: Produktbild.....	7
Abbildung 6: Produktbild.....	7
Abbildung 7: Phasenverschiebung von Pin A relativ zu Pin B.....	15
Abbildung 8: Schematische Darstellung der Verkabelung. Gleiche Farben entsprechen nicht zwingend gleichen Leitungen.....	21
Abbildung 9: Frontansicht.....	22
Abbildung 10: Rückansicht.....	22
Abbildung 11: Seitenansicht, links.....	22
Abbildung 12: Seitenansicht, rechts.....	22
Abbildung 13: Ansicht von unten.....	22
Abbildung 14: Draufsicht.....	22
Abbildung 15: Frontalansicht, eingeschaltet.....	23
Abbildung 16: Frontalansicht, ausgeschaltet.....	23

---

Abbildung 17: Schrägansicht, eingeschaltet.....	23
Abbildung 18: Schrägansicht, ausgeschaltet.....	23
Abbildung 19: Schrägansicht, eingeschaltet.....	23
Abbildung 20: Schrägansicht, ausgeschaltet.....	23

## 8 Quellenangaben

Alle Abbildungen ohne extra angegebene Quelle sind von mir selbst angefertigt und unterliegen damit dem Urheberrecht.

### 8.1 Abbildungsquellen

- Abb. 1 Quelle: Amazon, Produktbild  
URL: [https://images-na.ssl-images-amazon.com/images/I/61VONmTfZHL.\\_SL1240\\_.jpg](https://images-na.ssl-images-amazon.com/images/I/61VONmTfZHL._SL1240_.jpg)  
abgerufen am 08.03.2019
- Abb. 2 Quelle: Amazon, Produktbild  
URL: [https://images-na.ssl-images-amazon.com/images/I/91MSiH6Q7KL.\\_SL1500\\_.jpg](https://images-na.ssl-images-amazon.com/images/I/91MSiH6Q7KL._SL1500_.jpg)  
abgerufen am 08.03.2019
- Abb. 3 Quelle: Amazon, Produktbild  
URL: [https://images-na.ssl-images-amazon.com/images/I/715-EGYY7bL.\\_SL1200\\_.jpg](https://images-na.ssl-images-amazon.com/images/I/715-EGYY7bL._SL1200_.jpg)  
abgerufen am 08.03.2019
- Abb. 4 Quelle: Raspberry Pi Foundation, Produktbild  
URL: <https://www.raspberrypi.org/app/uploads/2017/05/PI-Zero-W-1-1620x1080.jpg>  
abgerufen am 08.03.2019
- Abb. 5 Quelle: Arduino.cc, Produktbild  
URL: [https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/520x330/604a3538c15e081937dbfb20aa60aad/a/0/a000066\\_featured\\_1.jpg](https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/520x330/604a3538c15e081937dbfb20aa60aad/a/0/a000066_featured_1.jpg)  
abgerufen am 08.03.2019
- Abb. 6 Quelle: Espressif Systems, Produktbild  
URL: <https://dl.espressif.com/dl/schematics/pictures/esp32-devkitc-v4-front.jpg>  
abgerufen am 08.03.2019
- Abb. 7 Quelle: Mark, „Sagsaw“: Two square waves in quadrature  
Veröffentlicht: Wikimedia, 2007  
URL: [https://upload.wikimedia.org/wikipedia/commons/6/68/Quadrature\\_Diagram.svg](https://upload.wikimedia.org/wikipedia/commons/6/68/Quadrature_Diagram.svg)

## 8.2 Literaturquellen

- I      Quelle: Amazon  
Produktbeschreibung  
URL: [https://www.amazon.de/Muse-M-196-DWT-Radio-Wecker-Senderspeicher/dp/B01DNPYMHK/ref=sr\\_1\\_1](https://www.amazon.de/Muse-M-196-DWT-Radio-Wecker-Senderspeicher/dp/B01DNPYMHK/ref=sr_1_1)  
abgerufen am 08.03.2019
- II     Quelle: Amazon  
Produktbeschreibung  
URL: [https://www.amazon.de/Pure-Evoke-H2-UKW-Radio-Küchen-Timer/dp/B01MCQPO8R/ref=sr\\_1\\_11](https://www.amazon.de/Pure-Evoke-H2-UKW-Radio-Küchen-Timer/dp/B01MCQPO8R/ref=sr_1_11)  
abgerufen am 08.03.2019
- III    Quelle: Amazon  
Produktbeschreibung  
URL: [https://www.amazon.de/FLOUREON-Stromversorgungsmodi-Helligkeit-Temperatur-Luftfeuchtigkeit/dp/B07FXP7VCY/ref=sr\\_1\\_5](https://www.amazon.de/FLOUREON-Stromversorgungsmodi-Helligkeit-Temperatur-Luftfeuchtigkeit/dp/B07FXP7VCY/ref=sr_1_5)  
abgerufen am 08.03.2019
- IV    Quelle: Raspberry Pi Foundation  
Produktdatenblatt  
URL: <https://www.raspberrypi.org/products/raspberry-pi-zero-w/>  
abgerufen am 08.03.2019
- V     Quelle: Arduino.cc  
Produktdatenblatt  
URL: <https://store.arduino.cc/arduino-uno-rev3>  
abgerufen am 08.03.2019
- VI    Quelle: Espressif Systems  
ESP32 Datasheet, V2.9, 28.02.2019  
URL: [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)  
abgerufen am 08.03.2019
- VII   Quelle: Brogle GmbH & Co. KG  
Sandra Brogle: Die elektrische Uhr  
URL: <https://www.broggle.de/ratgeber/uhren/quarzuhr/>  
abgerufen am 08.03.2019
- VIII   Quelle: Maxim Integrated  
DS3231 Datasheet, Rev. 10, 2015  
URL: <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>  
abgerufen am 08.03.2019

- 
- IX Quelle: hopf Elektronik GmbH  
B. Rega: Arbeitsweise und Genauigkeitsvergleich DCF77 und GPS Zeitempfänger  
URL: [https://hopf.com/dcf77-gps\\_de.html](https://hopf.com/dcf77-gps_de.html)  
abgerufen am 08.03.2019
- X Quelle: gsc-elektronic  
Gerhard Schmidt: Siebensegmentdecoder mit Anzeige, 2012  
URL: [http://www.gsc-elektronic.net/digitalelektronik/09\\_bcd\\_to\\_7.html](http://www.gsc-elektronic.net/digitalelektronik/09_bcd_to_7.html)  
abgerufen am 08.03.2019
- XI Quelle: Arduino.cc  
Tutorial Potentiometer  
URL: <https://www.arduino.cc/en/Tutorial/Potentiometer>  
abgerufen am 08.03.2019
- XII Quelle: Arduino.cc  
Dejan: How Rotary Encoder Works and How To Use It with Arduino, 2016  
URL: <https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/>  
abgerufen am 08.03.2019
- XIII Quelle: Arduino.cc  
Paul Badger: Capacitive Sensing Library  
URL: <https://playground.arduino.cc/Main/CapacitiveSensor?from=Main.CapSense>  
abgerufen am 08.03.2019
- XIV Quelle: DFRobot  
DFPlayer Mini Datenblatt, 15.08.2018  
URL: [https://www.dfrobot.com/wiki/index.php/DFPlayer\\_Mini\\_SKU:DFR0299](https://www.dfrobot.com/wiki/index.php/DFPlayer_Mini_SKU:DFR0299)  
abgerufen am 08.03.2019
- XV Quelle: Arch Linux Wiki  
Arduino, 05.01.2019  
URL: <https://wiki.archlinux.org/index.php/Arduino>  
abgerufen am 08.03.2019
- XVI Quelle: resistorguide  
Pull up resistor / Pull down resistor  
URL: [http://www.resistorguide.com/pull-up-resistor\\_pull-down-resistor/](http://www.resistorguide.com/pull-up-resistor_pull-down-resistor/)  
abgerufen am 08.03.2019
- XVII Quelle: Arduino Library  
Paul Badger und Paul Stoffregen: Capacitive Sensor Library, 21.02.2016  
URL: <https://github.com/arduino-libraries/CapacitiveSensor/>  
abgerufen am 08.03.2019

- XVIII Quelle: Eeveblog Forum  
Soldering to aluminium 2012  
URL: <https://www.eevblog.com/forum/chat/soldering-to-aluminium/>  
abgerufen am 08.03.2019
- XIX Quelle: Technische Universität Ilmenau  
Vorgehen Wasserfallmodell  
URL: <https://www.tu-ilmenau.de/sse/lehre-archiv/sommer-2014/softwareprojekt/vorgehensmodell-wasserfallmodell/>  
abgerufen am 09.03.2019
- XX Quelle: Wikipedia, 01.02.2019  
Erdung, 01.02.2019  
Referenziert aus: Schweizerischer Verband der Straßen- und Verkehrs fachleute: Erdungshandbuch.  
Regelwerk Technik Eisenbahn, Bern 2008  
URL: <https://de.wikipedia.org/wiki/Erdung>  
abgerufen am: 08.03.2019