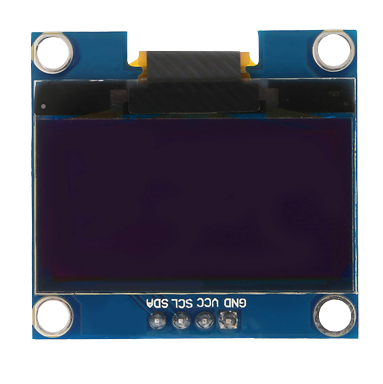
Ein Bild, das Uhr, Digitale Uhr, Schwarzweiß enthält.

Automatisch generierte BeschreibungEin Bild, das Symbol, Kreis, Logo, Grafiken enthält.

Automatisch generierte Beschreibung

Schachuhr

Verfasser/innen: David Kedra

Lukas Gregor

Betreuer: Dipl.-Ing. Kurt STADLER

Abteilung: Elektronik und technische Informatik

Schuljahr: 2023/24

Kurzfassung

Das vorliegende Projekt befasst sich mit der Entwicklung einer Schachuhr, die es Spielern ermöglicht, die Spielzeit jedes Spielers individuell einzustellen. Die Bedienung erfolgt über ein OLED-Display und einen Rotary Encoder. Die Spieler können die Zeit für ihr Spiel im Menü einstellen und deren aktuelle Spielzeit auf einem 7-Segment-Display verfolgen. Das System ermöglicht das Starten, Beenden und Pausieren von Spielen. Die Schachuhr funktioniert mit einem Arduino Mega.

**Inhaltsverzeichnis**

[Kurzfassung 1](#_Toc155541424)

[1 Projektplanung 3](#_Toc155541425)

[1.1 Stückliste 3](#_Toc155541426)

[1.2 I/O-Belegung 3](#_Toc155541427)

[2 Ergebnisse 4](#_Toc155541428)

[2.1 Schaubild 4](#_Toc155541429)

[3 Bedienungsanleitung 5](#_Toc155541430)

[4 Codedokumentation 6](#_Toc155541431)

[4.1 Lukas Gregor 6](#_Toc155541432)

[4.1.1 Drehgeber 6](#_Toc155541433)

[4.1.1.2 read\_rotary() 6](#_Toc155541434)

[4.1.1.3 click() 7](#_Toc155541435)

[4.1.2 OLED 7](#_Toc155541436)

[4.1.2.1 mode\_select() 7](#_Toc155541437)

[4.2 David Kedra 9](#_Toc155541438)

[4.2.1 7-Segment-Anzeige 9](#_Toc155541439)

[4.2.1.1 Segment Muster 9](#_Toc155541440)

[4.2.1.2 Segment Setup 9](#_Toc155541441)

[4.2.1.3 Display Segment 10](#_Toc155541442)

[4.2.1.4 Send Mux (Kommunikation mit MAX2719) 10](#_Toc155541443)

[5 Abbildungsverzeichnis 12](#_Toc155541444)

[6 Quellen 13](#_Toc155541445)

# Projektplanung

## Stückliste

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Eigenschaft | Stück | Anwendung |
| Anzeigemodul | 7-Segment mit MAX7219 | 1 | Anzeige der Zeit |
| Rotary Encoder | KY040 | 1 | Menü-Steuerung |
| OLED-Display | I2C, 128x64, 0,96 Zoll | 1 | Anzeige |
| Taster | NO, Rot | 1 | 1. Spieler |
| Taster | NO, Schwarz | 1 | 2. Spieler |
| Arduino | Nano | 1 | Prozessor |
| Widerstand | 100k | 2 | Pull-Down für Taster |

Tabelle 1 Stückliste

## I/O-Belegung

Alle Pins sind bezogen auf ein Arduino Mega.

**7-Segment Anzeige:**

5 (Data), 6 (CS), 7 (CLK)

**Taster**:

8 (Taster 0), 9 (Taster 1)

**Rotary Encoder**:

10 (CLK), 11 (DATA), 12 (SW)

**OLED-Display**:

20 (SDA), 21 (SCL)

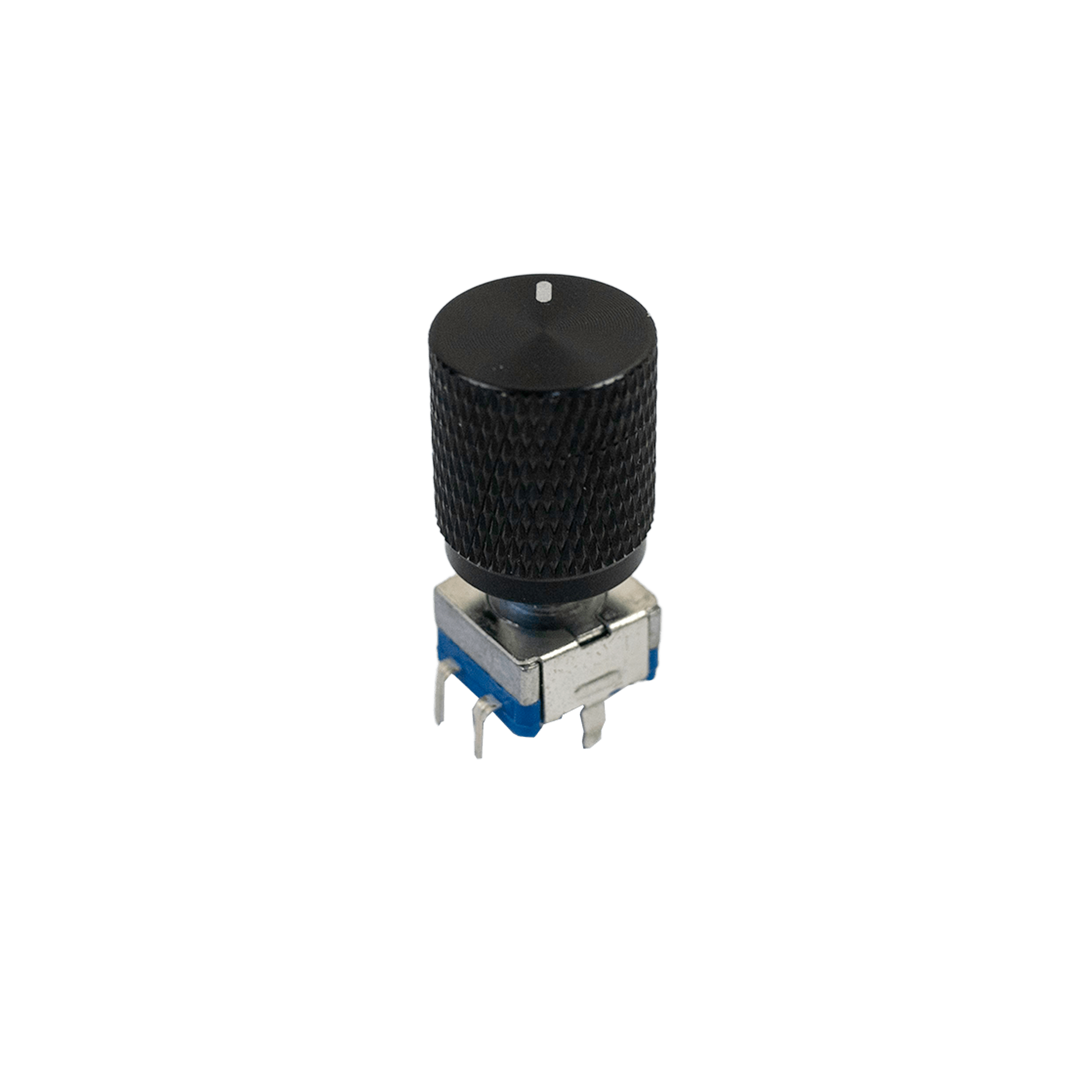
# Ergebnisse

## Schaubild

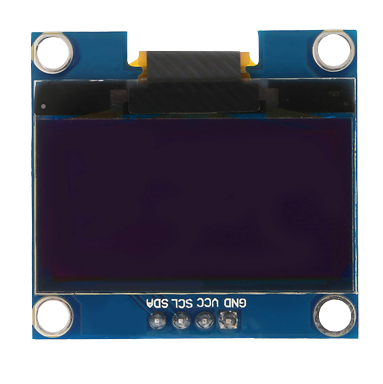
Rotary Encoder

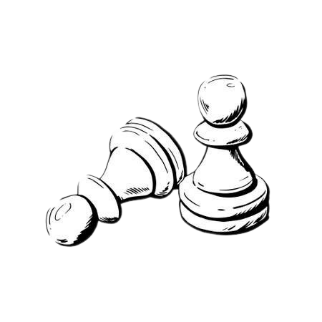
Taster  
Spieler 2

Taster  
Spieler 1



Display





Ein Bild, das Uhr, Digitale Uhr, Schwarzweiß enthält.

Automatisch generierte Beschreibung

Abbildung 1 Schachuhr Schaubild

8x 7 Segment Anzeigen

# Bedienungsanleitung

**Menüauswahl:**

Das OLED-Display zeigt das Hauptmenü an, in dem Sie die Einstellungen, wie die Zeiteinstellungen vornehmen können.  
Verwenden Sie den Rotary Encoder, um das Menü zu durchlaufen.  
Drücken Sie den Rotary Encoder, um eine Option auszuwählen.

**Zeiteinstellungen:**

Im Menü können Sie die Zeit für das Spiel pro Spieler einstellen. Die Standardzeit beträgt 10 Minuten oder 5 Minuten pro Spieler.  
Bei Auswahl der manuellen Einstellung verwenden Sie den Rotary Encoder, um die Minuten- und Sekundenwerte zu ändern.  
Drücken Sie den Rotary Encoder, um die Auswahl zu bestätigen.

**Spiel starten:**

Nachdem Sie die Zeiteinstellungen vorgenommen haben, beginnt die Zeit abzulaufen, sobald ein Spieler seinen Taster betätigt hat.

**Spielverlauf:**

Der Spieler, der nicht den Taster betätigt hat, beginnt mit seinem Zug.  
Die Zeit für beide Spieler wird auf dem 7-Segment-Display angezeigt: Die verbleibenden Minuten, gefolgt von einem Punkt und den verbleibenden Sekunden.

**Zug beenden:**

Jeder Spieler hat einen eigenen Taster. Um Ihren Zug zu beenden, drücken Sie Ihren Taster. Der Zug des anderen Spielers beginnt und seine Zeit fängt an abzulaufen.

**Spielende:**

**Zeitablauf:** Wenn die Zeit für einen Spieler abgelaufen ist, wird das Spiel beendet. Die Zeitanzeige des verlorenen Spielers beginnt zu blinken.

Um ein neues Spiel zu starten, betätigen Sie den Arduino-Reset-Taster.

# Codedokumentation

## Lukas Gregor

### Drehgeber

### read\_rotary()

int8\_t read\_rotary() {

static int8\_t rot\_enc\_table[] = { 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0 }; // Tabelle für erlaubte und unerlaubte Zustände

prevNextCode <<= 2;

if (digitalRead(DATA)) prevNextCode |= 0x02;

if (digitalRead(CLK)) prevNextCode |= 0x01;

prevNextCode &= 0x0f;

// Wenn gültig, dann als 16-Bit-Daten speichern.

if (rot\_enc\_table[prevNextCode]) {

store <<= 4;

store |= prevNextCode;

//if (store==0xd42b) return 1;

//if (store==0xe817) return -1;

if ((store & 0xff) == 0x2b) return -1; // Rückwärtsbewegung erkannt

if ((store & 0xff) == 0x17) return 1; // Vorwärtsbewegung erkannt

}

return 0;

}

Diese Unterfunktion[[1]](#endnote-2) ist für die Abfrage und Erkennung von Bewegungen zuständig. Er erkennt, wenn sich der Drehgeber um ein Inkrement bewegt. Mittels einer Wahrheitstabelle für erlaubte und unerlaubte Bewegungen kann das Unterprogramm Signale, die durch „bouncen“ entstehen können, eliminieren. Es wird abhängig von der Richtung, in die der Drehgeber gedreht wird, eine 1 oder eine -1 returned.

### click()

bool click() {

if (digitalRead(SW) == 0) {

delay(10);

display.invertDisplay(true);

while (digitalRead(SW) == 0) {

delay(10);

}

display.invertDisplay(false);

return true;

}

return false;

}

Das click() Unterprogramm besteht aus einem einfachen digitalRead, der erkennt wenn der Drehgeber gedrückt wird und gelangt anschließend in eine Schleife bis der Drehgeber losgelassen wird, um viele schnell aufeinanderfolgende Signale zu verhindern. Die Funktion display.invertDisplay() ; dient als visueller Indikator am OLED, dass der Drehgeber gedrückt wurde.

### OLED

### mode\_select()

int mode\_select() {

static int8\_t c, val;

while (!click()) {

if (val = read\_rotary()) {

c += val;

if (c == -1) { c = 2; }

c = c % 3;

Serial.println(c);

display.clearDisplay();

display.drawBitmap(0, 0, epd\_bitmap\_allArray[0], 128, 64, 1);

display.drawBitmap(27, 0, epd\_bitmap\_allArray[1], 74, 43, 1);

display.drawBitmap(27, 44, epd\_bitmap\_allArray[c + 2], 74, 21, 1);

display.display();

}

}

return c;

}

Das Unterprogramm wird in einer Schleife ausgeführt und aktualisiert bis zum nächsten Drehgeber-Druck das OLED mit den Bitmaps, die für die Modi-Anzeige nötig sind.

Im main loop() wird je nach Spieler-Zug und dem Pausen-Zustand die passende Bitmap mit display.drawBitmap(Bitmap\_Anfang\_X, Bitmap\_Anfang\_Y, epd\_bitmap\_allArray[Index], Breite, Höhe, Farben\_Invertierung); auf dem OLED angezeigt.

epd\_bitmap\_allArray[] ist eine Liste für alle Bitmaps um die Übersicht beizubehalten.

Die einzelnen Bitmaps werden aus der oled.h Datei importiert.

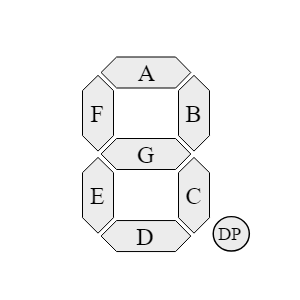


Abbildung 2 OLED Bitmap

## David Kedra

### 7-Segment-Anzeige

### Segment Muster

struct Segment {

uint8\_t numbers[10] = {

0b01111110, // 0

0b00110000, // 1

0b01101101, // 2

0b01111001, // 3

0b00110011, // 4

0b01011011, // 5

0b01011111, // 6

0b01110000, // 7

Abbildung 3 7-Segment

0b01111111, // 8

0b01111011, // 9

};

};

Die Ausgabe pro 7-Segment-Anzeige kann als 8-Bit-Muster veranschaulicht werden. Wenn ein Bit 1 ist, leuchtet der entsprechende Teil, wenn 0, leuchtet es nicht. Das erste Bit des 8-Bit-Musters ist der Punkt (*DP*). Die restlichen 7 Bits sind die Segmenten *A* bis *G*.

### Segment Setup

void MAX2719::segmentSetup() {

// display setup

sendMux(0x0B07); // Scan Limit Register

delay(10);

sendMux(0x0A0F); // Intensity (Helligkeit)

delay(10);

sendMux(0x0C01); // Shutdown Register

delay(10);

sendMux(0x0900); // Decode Mode

delay(10);

clearDisplay();

}

Im segmentSetup wird, wie laut Datenblatt[[2]](#endnote-3), der MAX2719 eingerichtet mit Übergabe von bestimmten Werten. Die ersten zwei Ziffern der Hexadezimalzahl bestimmen den Modus, die nächsten zwei den übergebenen Wert.

Helligkeit als Beispiel: sendMux(0x0A0F);

0A steht für den Modus „Intensity“ und 0F steht für die maximale Helligkeit.

### Display Segment

void MAX2719::displaySegment(int pos, int data) {

data += (9 - pos) \* 0x0100;

sendMux(data);

}

Hier wird an der Position pos data angezeigt. Der Parameter data ist das anzuzeigende Muster in 8 Bit.

0x01.. steht für die 7-Segment-Anzeige auf der letzten/rechten Position. 0x08.. steht für die 7-Segment-Anzeige auf der ersten/linken Position. Es wird (9 - pos) verwendet, um dies umzukehren. Multipliziert man dies mit 0x0100 wird die korrekte Position eingetragen. Dies wird nun zu data addiert. .

Da data ein 8-Bit-Muster ist, das in Hexadezimalzahl durch zwei Ziffern repräsentiert wird, werden bei der Addition die dritte und vierte Ziffer überschrieben und data gilt als übergebener Wert.

### Send Mux (Kommunikation mit MAX2719)

void MAX2719::sendMux(unsigned int data) {

digitalWrite(CS, LOW);

bool zw;

unsigned int maske = 0x8000;

for (int i = 0; i < 16; i++) {

digitalWrite(CLK, LOW);

zw = (data & maske) == maske;

maske = maske >> 1;

digitalWrite(DATA, zw);

digitalWrite(CLK, HIGH);

}

digitalWrite(CS, HIGH);

}

Die Funktion sendMux implemetiert die Kommunikation mit einem MAX2719. Der Chip Select-Pin (CS) wird auf LOW gesetzt, um die Kommunikation mit dem MAX2719 zu starten. Die Maske maske (binär: 1000000000000000) dient dazu, um bei dem 16-Bit-Parameter data jeden Bit einzeln zu übertragen. In der Schleife wird jedes einzelne Bit nacheinander übertragen, indem das zur Maske bestimmte Bit entnommen wird und der Daten-Pin DATA auf diesen Wert gesetzt wird. Nach jeder Übergabe von DATA wird die Clock (CLK) von LOW zu HIGH geswitched, um das Bit zu übertragen. Man braucht dazu kein delay, da der MAX2719 schneller ist als die Clock des Arduinos. Am Ende wird der Chip Select-Pin (CS) auf HIGH gesetzt, um die Kommunikation mit dem MAX2719 zu beenden.

### Timer

void timerSetup() {

// Timer1 im CTC-Modus auf 1 Sekunde einstellen

noInterrupts(); // alle Interrupts abschalten cli()

TCCR1A = 0; // Registereintrag löschen

TCCR1B = 0; // Registereintrag löschen

TCCR1B |= (1 << CS12); // 256 als Prescale-Wert spezifizieren

TCCR1B |= (1 << WGM12); // Setzt das Bit für den CTC-Mode

OCR1A = 62499; // Auf Endwert setzen 62500 Impulse zählen = 1 Sekunde

TIMSK1 |= (1 << OCIE1A); // Schaltet den Compare-Interrupt ein

interrupts(); // Alle Interrupts scharf schalten sei()

}

Dies erstellt einen Interrupt-Timer TIMER1, welcher jede 62500 Impulse, also jede Sekunde auslöst.

ISR(TIMER1\_COMPA\_vect) {

max2719.timerLogic();

}

Diese Funktion wird bei jedem Auslösen des TIMER1 ausgeführt.

Siehe Quelle[[3]](#endnote-4).

### Anzeige der Zeit

void MAX2719::displayTime(bool leftPos, int seconds) {

int minutes = (seconds / 60) % 100; // maximale Minutenanzahl = 99

int remainingSeconds = seconds % 60;

int result = minutes \* 100 + remainingSeconds; // Anzeige = [minutes].[seconds]

displayInt(leftPos, result);

}

Da die übergebliebene Zeit in Sekunden gespeichert wird, muss dies in Minuten und restliche Sekunden umgeformt werden. Nun muss dies in ein Integer gespeichert werden, wo die Einer- und Zehner-Stellen die Sekunden darstellen und die restlichen, maximal 2 weiteren Ziffern die Minuten darstellen.

void MAX2719::displayInt(bool leftPos, int num) {

...

...

// Anzeige

for (int i = 0; i < 4; i++) {

int displayPos = i + 1; // displayPos: 1 steht für die erste MAX2719-Anz, 8 für die letzte

if (leftPos) {

displayPos += maxDigits; // Für linke Seite

}

if (i < count) {

int data = segment.numbers[digits[i]]; // Ziffer

if (i == 0 && count == 3) {

data += 0b10000000;

} else if (i == 1 && count == 4) {

data += 0b10000000;

}

displaySegment(displayPos, data);

} else {

int data = 0; // clear

displaySegment(displayPos, data);

}

}

}

Diese Funktion verwendet den Integer-Wert result von der Funktion displayTime, wandelt diesen in einzelne Ziffern um und speichert dies in das Array digits. Es wird überprüft, ob ein Punkt platziert werden soll und wenn ja, an der richtigen Stelle (Nach den Minuten) platziert, indem zu data 0b10000000 addiert wird, was auf einen Punkt am 7-Segment-Display hindeutet.

# Abbildungsverzeichnis

[Abbildung 1 Schachuhr Schaubild 4](https://d.docs.live.net/0bc58d46afe5b98c/HWE%20Schachuhr.docx#_Toc155546404)

[Abbildung 2 OLED Bitmap 8](https://d.docs.live.net/0bc58d46afe5b98c/HWE%20Schachuhr.docx#_Toc155546405)

[Abbildung 3 7-Segment 9](https://d.docs.live.net/0bc58d46afe5b98c/HWE%20Schachuhr.docx#_Toc155546406)

# Quellen

1. Drehgeber debounce Code: <https://www.best-microcontroller-projects.com/rotary-encoder.html#Taming_Noisy_Rotary_Encoders> [↑](#endnote-ref-2)
2. Datenblatt MAX7219: <https://www.analog.com/media/en/technical-documentation/data-sheets/max7219-max7221.pdf> [↑](#endnote-ref-3)
3. Timer-Interrupt Dokumentation von Dipl.-Ing. Kurt Stadler <https://moodle.htl-donaustadt.at/pluginfile.php/118835/mod_resource/content/1/Dig_Temp_Doku_Timer.pdf> [↑](#endnote-ref-4)