# Dokumentation zum Weckerprojekt

Moritz Nörenberg und Jacob Ueltzen

# Inhaltsverzeichnis

# 1 Einführung

Im folgenden soll unser Weckerprojekt beschrieben werden, welches auf dem MSP-Education System umgesetzt werden sollte. Dabei sind folgende Vorgaben zu beachten gewesen:

1: Weckuhr mit Anzeige von Uhrzeit, Datum, Wochentag und Alarmzeit über das LCD

2: Eingabe und Justage von Uhrzeit, Alarmzeit und Datum

3: Akustischer Weckalarm aus unterschiedlichen Tönen

4: Nutzung der 4x4 LED Matrix zur Anzeige der Uhrzeit als Laufschrift (Stunden : Minuten)

Ich werde versuchen den Verlauf unserer Arbeit in dieser Dokumentation zu reproduzieren und die Funktionen und Module in der Reihenfolge zu erläutern, in welcher sie von uns implementiert wurden, um für den Betrachter verständlich zu machen, welchen Problemen wir bei unserer Arbeit begegnet sind und wie wir diese gelöst haben. Außerdem hoffe ich so eine bessere Verständlichkeit für unseren Quelltext vermitteln zu können.

# 2 Uhrzeitanzeige

Begonnen haben wir zunächst damit, dass wir eine angezeigte Uhrzeit auf dem LCD des Education Systems anzeigen wollten. Dabei haben wir uns dafür entschieden folgenden Variablentyp neu anzulegen:

```
1       typedef struct
2       {
3       uint8_t sec;     //
4       uint8_t min;
5       uint8_t hour;
6       uint8_t day;
7       uint8_t mon;
8       uint16_t year;
9       } time_t;
10
11      time_t time;
```

Dabei soll in den einzelnen Elementen jeweils der entsprechende Teil der Uhrzeit gespeichert werden. time.hour soll also nur den Wert der Stunde haben, um 21:10 Uhr also den Wert 21.
Als nächstes brauchten wir eine Funktion, die uns die in time gespeicherten Werte auf dem LCD anzeigen kann. Für das ansteuern des LCD liegt im zur Vorlesung Mikrorechnerarchitekturen gehörigen Repository von Robert Fromm eine Header- und die zugehörige Source-Datei, die lcd.h und lcd.c, welche wir zu unserem Projekt hinzugefügt haben. Damit erhielten wir nun Zugriff auf sämtliche dort deklarierten und definierten Funktionen. Darunter auch die Funktion `lcd_put_char()`, welche in der Lage ist einzelne character auf dem LCD darzustellen. Da unsere Variable time nun aber keine character Variablen speichert brauchten wir zusätzlich noch eine Funktion, welche die gespeicherte Zeit so umrechnet, dass sie mit der gegeben Funktion auf dem LCD ausgegeben werden kann.

```
void outputtime()
{
lcd_put_char((time.hour - (time.hour % 10)) / 10 + 48);
lcd_put_char((time.hour % 10) + 48);
lcd_write(":");
lcd_put_char((time.min - (time.min % 10)) / 10 + 48);
lcd_put_char((time.min % 10) + 48);
lcd_write(":");
lcd_put_char((time.sec - (time.sec % 10)) / 10 + 48);
lcd_put_char((time.sec % 10) + 48);
}
```

Dabei berechnet die erste Zeile der Funktion den Zehner der Stunde und gibt diesen aus, die zweite Zeile berechnet den Einer der Stunde und gibt diesen aus und die Dritte Zeile gibt einen Doppelpunkt als Trennzeichen aus. Die 48 am Ende jeder Zeile sorgt dafür, dass auch die richtigen Zeichen ausgegeben werden. 48 ist in ASCII eine 0, also wird auf jeden errechneten wert diese 48 addiert um das entsprechende Zeichen zum Wert auszugeben. Auf die selbe Art wird noch mit Minuten und Sekunden verfahren. Ähnlich sind wir dann auch mit dem Datum verfahren, bis auf die Berechnung des Jahres, die mit dieser Methode recht umständlich und lang geworden wäre. Das Jahr haben wir daher in ein Array geschrieben, welches wir dann über die Funktion `lcd_write()` ausgeben können, welche ebenfalls in der lcd.c definiert ist.

```
  char Jahr[5];
  snprintf(Jahr, sizeof(Jahr), "%d", time.year);
  lcd_write(Jahr);
```

Damit ist nun die Ausgabe von Uhrzeit und Datum möglich, auch wenn sich diese noch nicht aktualisiert und nur über den Quellcode vordefinierte Werte annehmen kann. (Vgl. main.c line 29-34)

## 3 Aktualisierung der Uhrzeit

Für die sekündliche Aktualisierung der Uhrzeit war es nun nötig einen sekündlich auslösenden Interrupt zu generieren. Dafür haben wir den Watchdog-timer so initialisiert, dass er viertelsekündlich einen Interrupt auslöst.

```
  WDTCTL = WDTPW + WDTTMSEL + WDTCNTCL + WDTSSEL;
  IE1 |= WDTIE;
```

Bei jedem Auslösen des Interrupts wird unser Variablenelement time.sec um eins nach oben gezählt und außerdem für spätere Anwendungen auch das `BIT0` in die Variable `second_flag` geschrieben. Diese heißt nicht so, weil es in einer Form die zweite ist, sondern weil sie sekündlich auslöst.

```
#pragma vector=WDT_VECTOR       //Interrupt Vektor, der die Struktur tim
__interrupt
void WDT_ISR()
{
        time.sec++;
        second_flag = BIT0;

   __low_power_mode_off_on_exit();
}
```

Damit kann nun die Sekundenanzeige im Sekundentakt nach oben zählen. Allerdings bisher nur ASCII Zeichen und selbst wenn man bei 0 anfangen lässt würde der Counter über die 10 hinaus zählen und die zugehörigen ASCII Zeichen auf dem LCD ausgeben. Dafür haben wir dann die Funktion `timeCorrection` geschrieben. Hier nur ein Auszug daraus:

```
void timeCorrection()
{  //overflow handling aller Zeiteinheiten
if (time.sec > 59)
{
time.sec = 0;
time.min++;

if (time.min > 59)
{
time.min = 0;
time.hour++;
...
}
```

Diese Funktion geht in ähnlicher Form durch alle Elemente der Variable time. Bei den Monaten ist es dann etwas komplizierter da die Monate unterschiedlich viele Tage haben können. Außerdem befindet sich in dieser Funktion auch noch die Berechnung der Schaltjahre:

```
...
 else if (time.mon == 2)
 {
   if (time.year % 4 == 0 && time.year % 100 != 0
    || time.year % 400 == 0)
  {
    if (time.day > 29)
    {
       time.mon++;
       time.day = 1;
    }
  }
  else
  {
    if (time.day > 28)
    {
       time.mon++;
       time.day = 1;
    }
  }
}
...
```

Da es nun nur noch möglich ist existierende Daten und Uhrzeiten auszugeben und nicht Daten wie zum Beispiel den 30.Februar, konnten wir uns nun als Nächstes daran machen eine Berechnung für die Wochentage anzustellen. Dafür haben wir die von Gauß aufgestellte Formel zur Berechnung von Wochentagen zur Hilfe genommen, deren Gültigkeitsbereich zwischen dem 15.10.1582, dem Tag der Einführung des Gregorianischen Kalenders und dem Jahr 3000 liegt.
Die Folgende Funktion berechnet die Laufvariable `uint8_t weekdayi` , welche Werte zwischen 0 und 6 annehmen kann, die einen linearen Zusammenhang zum Wochentag haben. Ist `weekdayi` gleich 0, so ist der errechnete Wochentag Sonntag, bei einer Eins ein Montag usw. Die Ausgabe der Wochentage haben wir über eine switch-case Anweisung realisiert. Davon aber hier ebenfalls nur einen Auszug.

```
1  void weekdayinit()
2  {//Berechnung von weekdayi mithilfe
3  uint8_t h; //der Gauss'schen Wochentagsformel
4  uint16_t k;
5
6  if (time.mon <= 2)
7  {
8    h = time.mon + 12;
9    k = time.year - 1;
10 }
11 else
12 {
13   h = time.mon;
14   k = time.year;
15 }
16
17 weekdayi = (time.day + 2 * h + (3 * h + 3) / 5 + k + k / 4 - k / 100
18 + k / 400 + 1) % 7;              //Wochentagsberechnung
19
20 }
21
22
23 void outputday() //Ausgabe des jeweiligen Wochentags
24 {
25   lcd_gotoxy(0, 1);
26   switch (weekdayi)
27   {
28   case 0:
29     lcd_write("SUN,");
30     break;
31   case 1:
32     lcd_write("MON,");
33     break;
34   case 2:
35     lcd_write("TUE,");
36     break;
37   ...
38 }
```

Damit ist unser Wecker nun in der Lage zu einem Datum den richtigen Wochentag anzuzeigen und dabei die Zeit zu zählen. Wenn also die Zeit die richtige ist, wäre der Wecker fertig. Das einzige, was nun also noch fehlt, um die Zeitfunktionen des Weckers zu vollenden ist die Möglichkeit, Uhrzeit und Datum am Wecker selbst einstellen zu können.

# 4 Initialisierung des Weckers

Wir haben zu Beginn die einfachste Möglichkeit gewählt Uhrzeit und Datum des Weckers einzustellen, die uns einfiel. Diese beinhaltete jedoch keine Möglichkeit innerhalb des Einstellungsmenüs wieder einen Schritt zurück zu gehen. Hatte man zum Beispiel die Stunden bereits eingestellt und war weiter gegangen zu den Minuten, so gab es keine Möglichkeit wieder zu den Stunden zurückzukehren. Eigentlich haben wir dieses Problem zu einem anderen Zeitpunkt gelöst, da wir aber bei der Lösung diese Problems eine Menge Code verändert und gelöscht haben werde ich jetzt nur noch auf die letztendliche Lösung eingehen und einige Schritte in der Entstehung des Weckers auslassen.

## 4.1 Einstellung der Uhrzeit

Für die Einstellung der Uhrzeit haben wir eine Funktion geschrieben, welche die momentan ausgegebene Uhrzeit verändern kann. Dafür wird eine Zählvariable eingeführt, die vom Drehencoder hoch oder runter gezählt werden kann. Also mussten wir zunächst erst den Drehencoder interruptfähig machen und eine hoch- und runterzähl Unterscheidung beifügen. Da wir das Grundprinzip für diese Funktionalität aber aus Robert Fromms GitHub Beispiel entnommen haben werde ich hier nicht tiefer darauf eingehen. Mithilfe des Drehencoders ist es uns nun aber möglich, dass wir über kurze if-schleifen eine Überlaufabfrage machen um keine unerlaubten Zahlen zu erreichen oder auch keine anderen ASCII Zeichen über das LCD Ausgegeben werden können. Hier ein Beispiel:

```
1  if (time.hour == 20)//Zehner der Stunden = 20
2    {//Also noch max. 3 moeglich, da 24 nicht erlaubt ist
3      if (a > 51)//51 in ASCII entsprich 3
4      {
5        a = 48;//48 entspricht 0
6      }
7      if (a < 48)
8      {
9        a = 51;
10     }
11   }
12   else//Zehner der Stunden nicht 20
13   {//Voller Zahlenumfang erlaubt
14     if (a > 57)//entspricht 9
15     {
16       a = 48;
17     }
18     if (a < 48)
19     {
20       a = 57;
21     }
22   }
```

Dabei wird unsere Zählvariable a dauerhaft auf dem LCD ausgegeben, damit der Benutzer des Weckers auch sehen kann, was er einstellt. Ist der Nutzer fertig mit dem Einstellen eines Digits und drückt den Weiter- oder Zurückbutton. Dann wird der aktuelle Wert von `a` als Dezimal-Ziffer umgerechnet im zugehörigen Variablenelement

gespeichert. Daraufhin wird `a` wieder zu 0 gesetzt und der Cursor je nach Eingabe des Nutzers um eine Stelle nach vorn oder nach hinten verschoben.

```
1  if (button_flag & BIT0)            // Weiter-Button
2  {
3    button_flag &= ~BIT0;
4    time.hour = time.hour + (a - 48);
5    a = 48;
6    setupTimeState = setupTimeStateTenMin;
7  }
8  if (button_flag & BIT1)            // Zurueck-Button
9  {
10   button_flag &= ~BIT1;
11   time.hour = time.hour + (a - 48);
12   a = 48;
13   setupTimeState = setupTimeStateTenHour;
14 }
```

Dieser Ablauf wird für jedes einzustellende Digit wiederholt und alles zusammen steht in einer switch-case Anweisung, die zwischen den verschiedenen Zuständen von `setupTimeState` unterscheidet und somit die Navigation innerhalb des Menüs ermöglicht.

## 4.2 Einstellung des Datums

Bei der Einstellung des Datums sind wir auf die selbe Weise vorgegangen wie bisher bei der Uhrzeit. Da wir uns bei der Einstellung des Datums in einem anderen Menüpunkt befinden und dieser auch unabhängig von der Uhrzeiteinstellung sein soll, haben wir das alles in eine weitere Funktion geschrieben. Diese heißt `void setupdate()` und macht im Grunde das Selbe wie die eben beschriebene Funktion. Deshalb werde ich hier nicht näher darauf eingehen, da dies nicht zu weiterem Verständnis der Funktionalität und dem Aufbau des Programmes beitragen würde.

## 4.3 (Neu)start des Weckers

Bei erstmaligen Starten des Programms oder bei Neustart des Weckers, ob über den auf dem Education System verbauten Reset Button oder durch Trennen der Stromverbindung, wird über eine Funktion namens `void startupscreen()`, die nur einmalig bei Neustart des Programms ausgeführt wird, die Initialisierung durchlaufen. Das bedeutet, dass der Benutzer auf dem LCD dazu aufgefordert wird die aktuelle Uhrzeit einzugeben und darauf hin die im Kapitel 4.1 beschriebene Funktion aufgerufen wird. Nachdem der Benutzer dann die Zeit eingegeben hat kann er durch bestätigen zur Einstellung des Datums kommen. Hierfür wird die im Kapitel 4.2 beschriebene Funktion aufgerufen. Wenn die Einstellung der Zeit abgeschlossen ist und der Nutzer dies bestätigt wird noch die Funktion zur Wochentagsberechnung ausgeführt und auf dem LCD erscheinen dann die Worte "Initialisierung abgeschlossen". Damit ist der Wecker vollständig initialisiert und wechselt automatisch in den Uhrzeit Bildschirm und beginnt damit die Uhrzeit zu zählen und anzuzeigen.
Da die gerade beschriebene Funktion `void startupscreen()`, nur aus Aufrufen bereits bekannter Funktionen besteht werde ich hier kein Codebeispiel dazu zeigen.

# 5 Alarm

Da nun die Grundfunktionen eines Weckers gegeben sind war es an der Zeit sich um zusätzliche Funktionen zu kümmern. Der dabei von uns zuerst bearbeitete Punkt war die Weckfunktion.
Da wir hier vieles gleichzeitig und auch teilweise Dinge nicht so benutzt haben, wie wir sie uhrsprünglich geschrieben haben werde ich für diesen Teil davon Abweichen den Code chronologisch zu erläutern, sondern hier die einzelnen Funktionalitäten der Weckfunktion erklären. Dadurch kann es dazu kommen, dass in einigen Beispielen Referenzen zu Funktionen stehen, die bisher noch nicht bekannt sind. Dies lässt sich leider nicht vermeiden.

## 5.1 Einstellen der Alarmzeit

Für das Einstellen der Weckzeit brauchten wir nun lediglich einen weiteren Menüpunkt und konnten dann wieder das Grundprinzip der vorangehend erläuterten Funktion zur Zeiteinstellung nutzen. Da diese Vorgehensweisen aber bereits bekannt sind werde ich nicht weiter darauf eingehen.
Zusätzlich haben wir uns auch eine weitere Variable angelegt, welche wieder die Uhrzeitvariablen enthält und eine weiteres Variablenelement namens .enable. Von dem neu definierten Typ gibt es die Variablen alarm und alarmold. Den Grund dafür werde ich später erläutern.
Da unser Vorgehen mit dem neuen Typ ähnlich dem Vorherigen ist, werde ich auch hier wieder nicht darauf eingehen.
Weiterhin war es jedoch auch nötig dem Nutzer die Möglichkeit zu geben, sich nicht wecken zu lassen. Dafür haben wir das Variablenelement .enable eingeführt, die grundsätzlich nur zwei Zustände haben kann. 1 zum Einschalten des Weckers, was auf dem LCD als ”EIN” ausgegeben wird und 0, um die Weckfunktion auszuschalten, was auf dem Display des Weckers als ”AUS” angezeigt wird. Der Zustand von .enable wird vom Nutzer auf die selbe Art eingestellt, wie die Einstellung der Weckzeit vorher (Drehencoder).

Um nun aber auch einen Weckvorgang zu realisieren brauchten wir noch etwas, das die aktuelle Zeit mit den in `alarm` gespeicherten und vom Nutzer eingegebenen Werten vergleicht. Dafür haben wir folgende if-Anweisung so in unser Hauptprogramm geschrieben, dass sie dauerhaft wieder aufgerufen wird.

```
1       if (time.hour == alarm.hour && time.min == alarm.min
2      && alarm.sec == time.sec && alarm.enable == 1)
3      {
4        menuState = menuState_WAKEUP;
5      }
```

Diese Schleife vergleicht alle einzelnen Elemente der Zeit mit denen, die im Alarm eingestellt worden sind. Dabei wären die Sekunden eigentlich gar nicht von Nöten gewesen, da die Weckzeit nur auf volle Minuten gestellt werden kann. Dennoch vergleichen wir diese mit, da wir somit das Problem umgehen können, dass der Wecker eine ganze Minute lang durch klingeln würde, ohne das der Nutzer in der Lage wäre das Wecken zu unterbrechen. Der Wert von `alarm.sec` kann nicht vom Nutzer geändert werden und ist 0 by default. Somit löst der Alarm nur zur vollen Minute der eingestellten Weckzeit aus und der Kopf der if-Schleife ist auch nur diese eine Sekunde lang wahr. Würde der Nutzer versuchen innerhalb dieser Sekunde den Alarm wieder abzustellen

könnte es zu Fehlern kommen und der Abstellknopf müsste ein weiteres Mal gedrückt werden.
Die Zuweisung im Rumpf der Schleife versetzt den Counter unseres Menü-Zählers in Alarmzustand. Das bedeutet, dass sich unsere große switch-case Anweisung in den Zustand `WAKEUP` begibt. Dieser wird im folgenden Kapitel erläutert.

## 5.2 Abstellen des Alarms

Zum Abstellen des Alarms bedarf es eines einzelnen Knopfdruckes. Durch unsere switch-case Anweisung können wir dies ganz einfach dadurch realisieren, dass wir die Abbruchbedingung in den Kopf einer if-Schleife schreiben und in deren Rumpf wiederum den neuen Zustand für das menuState definieren.
Die Funktion `timeCorrection()` muss mit während des Weckvorgangs ausgeführt werde, da der Wecker selbst sonst im Hintergrund nicht weiter zählen würde.

## 5.3 Klingeln des Weckers

Um den Wecker klingeln zu lassen muss der Nutzer also eine valide Alarmzeit eingegeben haben und das alarm.enable bit auf eins gesetzt haben. Wenn dies geschehen ist, ist auch auf dem Hauptbildschirm ein ”A” zu sehen, welches den Nutzer darauf hin weißt, dass ein Alarm aktiv ist.

Die Tonausgabe wurde mittels des CCR2 Registers und Timer A realisiert. Hierfür konfigurierten wir Timer A so, dass die Auxillary clock von Timer A verwendet wird. Außerdem haben wir uns für den kontinuierlichen Modus des Hochzählens entschieden. Verscheidene Töne haben wir mit verschiedenen Einträgen in das CCR2 Register realisiert. Diese werden durch die Variable toneccr in das CCR2 Register geschrieben. Zunächst bei der Initialisierung vor der while-Schleife im Hauptprogramm, danach wird viertelsekündlich ein anderer Wert in das CCR2 Register eingetragen. Das geschieht in der Interruptservicesoutine für den Watchdogtimer. Abhängig von der dort hochzählenden clock-Variable wird einer von vier verschiedenen Werten in das CCR2 Register geschrieben. Die Werte im CCR2 Register bestimmen, nach wie vielen Auxillary Taktzyklen der Lautsprecherausgang getoggelt werden soll. Somit werden Rechtecksignale verschiedener Frequenz erzeugt.

```
1       case 4:  // CCR2 Interrupt
2        CCR2 += toneccr;
3        if (clock == 0)
4        {
5          toneccr = 30;
6        }
7        else if (clock == 1)
8        {
9          toneccr = 37;
10       }
11       else if (clock == 2)
12       {
13         toneccr = 52;
14       }
15       else if (clock == 3)
16       {
```

```
17          toneccr = 74;
18        }
19        break;
```

# 6 Die LED-Matrix

Ein weiteres Feature des Weckers, welches laut den Vorgaben programmiert werden muss ist die LED-Matrix. Zum Ansteuern der Matrix haben wir uns die von Robert Fromm geschriebene `matrix.c` und die dazugehörige `matrix.h` in unser Projekt eingebunden. Dort sind unter anderem Funktionen wie die `matrix_init()`, welche zur Benutzung der LED-Matrix zwangsläufig aufgerufen werden muss. Auch einige andere essentielle Funktionen sind in diesen Dateien definiert, die in unserem Projekt Verwendung finden. Diese werde ich jedoch nicht alle einzeln erläutern.

Zu Beginn haben wir eine Funktion geschrieben, welche die komplette Matrix zum leuchten bringen kann und eine, welche alle LED's der Matrix wieder ausschaltet. Benutzt haben wir davon letztendlich nur die zum ausschalten, aber behalten haben wir die andere für Verwendung zu späteren Zeitpunkten trotzdem. Im Beispiel ist nun die Verwendete Funktion zum Ausschalten der Matrix zu sehen:

```
1     void matrix_clear()
2     {
3       matrix_write_row(0, 0b0000);
4       matrix_write_row(1, 0b0000);
5       matrix_write_row(2, 0b0000);
6       matrix_write_row(3, 0b0000);
7     }
```

Diese Funktion tut nichts weiter, als dass sie alle Elemente der Matrix mit Nullen beschreibt, also alle LED's ausschaltet.
Die Funktion zum Einschalten der Matrix sieht dieser sehr ähnlich, bis auf, dass in den Klammern als zweiter Wert überall eine binäre 15, also vier Einsen stehen um die einzelnen LED's der Matrix einzuschalten.

Für das Lauflicht auf der Matrix ist es wichtig, dass die einzelnen anzuzeigenden Zeichen irgendwo hinterlegt sind. Da wir nur die Zeit anzeigen wollen brauchten wir die Zahlen von 0 bis 9. Wie diese auf der Matrix aussehen sollen ist im folgenden Vektor hinterlegt.

```
1     uint8_t zahlen[10][3] = {
2         { 0b1111, 0b1001, 0b1111 },     //0
3         { 0b1010, 0b1111, 0b1000 },     //1
4         { 0b1001, 0b1101, 0b1011 },     //2
5         { 0b1001, 0b1011, 0b1111 },     //3
6         { 0b0111, 0b0100, 0b1110 },     //4
7         { 0b1011, 0b1011, 0b1101 },     //5
8         { 0b1111, 0b1011, 0b1101 },     //6
9         { 0b0001, 0b0101, 0b1111 },     //7
10        { 0b1111, 0b1011, 0b1111 },     //8
11        { 0b1011, 0b1011, 0b1111 }     //9
12    };
```

Dabei bilden immer drei der Binärzahlen eine Ziffer. Jede der Zeilen steht dabei für eine Spalte auf der Matrix. Bildlich vorstellen kann man sich das am besten am Beispiel der Null. Da hat der erste Eintrag vier Einsen, in der ersten Spalte der Matrix sind also alle LED's an. Der zweite Eintrag besteht aus zwei Nullen umgeben von je einer Eins, daraus resultiert in der zweiten Spalte der Matrix, dass die obere und die untere LED an sind, die beiden dazwischen aus. Der dritte Eintrag ist wieder gleich dem ersten, es entsteht wieder eine voll eingeschaltete Spalte. Das hintereinander ergibt das Abbild einer Null in den ersten drei Spalten der Matrix.
Damit sind wir nun in der Lage eine Zahl zur Zeit in der Matrix anzuzeigen, jedoch haben wir noch nicht das geforderte Lauflicht. Dafür haben wir uns einen Vektor mit 20 Einträgen definiert, der in einer Funktion immer mit der momentanen Zeit und denn nötigen Trenn- und Leerzeichen beschrieben wird. Dies sieht wie folgt aus:

```
1  void array_input()
2  { ...
3    output[3] = 0b0000;
4    output[4] = zahlen[(time.hour - (time.hour % 10)) / 10][0];
5    output[5] = zahlen[(time.hour - (time.hour % 10)) / 10][1];
6    output[6] = zahlen[(time.hour - (time.hour % 10)) / 10][2];
7    output[7] = 0b0000;
8    output[8] = zahlen[time.hour % 10][0];
9    output[9] = zahlen[time.hour % 10][1];
10   output[10] = zahlen[time.hour % 10][2];
11   output[11] = 0b0000;
12   output[12] = 0b1010;
13   ...
14 }
```

output[] ist der bereits genannte Vektor. Seine ersten vier Elemente werden nur mit Nullen beschrieben, damit die Uhrzeit "reinlaufen"kann und nicht bei Aufruf der Funktion bereits die erste Zahl auf der Matrix steht. Davon ist hier jedoch nur noch das vierte Element zu sehen.
Die Zeilen 4 bis 6 des Codebeispiels errechnen die Zehner der Stunde und verlangen den Wert des Vektors Zahlen in Abhängigkeit der errechneten Stunde. Das zweite Argument des Vektors zahlen ist dabei für die drei benötigten Spalten verantwortlich, die benötigt werden, um eine Zahl darzustellen.
In Zeile 7 ist dann eine Leerzeile als Trennung zwischen den Zahlen eingefügt, dann wiederholt sich das ganze mit den Einern der Stunde. In Zeile 12 ist ein Doppelpunkt als Trennung zwischen Stunden und Minuten eingefügt. Identisch wird mit den Minuten verfahren.

Für das Lauflicht ist nun die Grundlage gelegt. Ein Vektor, der die gesamte Uhrzeit so enthält, inklusive der Trenn- und Leerzeichen, die benötigt werden um die Zeit lesen zu können. Da die Matrix aber nur vier Elemente breit ist und wir 20 haben kommt nun das Laufen ins Spiel. Dafür haben wir eine weitere Funktion geschrieben, die dafür sorgt, dass immer nur vier Elemente unseres Vektors `output[]` auf der Matrix ausgibt:

```
1  void array_output()
2  {
3    if (k > 20)
4    {
5      k = 0;
```

```
  }
  array_input();
  matrix_write_row(3, output[k + 3]);
  matrix_write_row(2, output[k + 2]);
  matrix_write_row(1, output[k + 1]);
  matrix_write_row(0, output[k]);
  k++;
}
```

Dabei werden alle vier Spalten der Matrix beschrieben und zwar mit den Werten, die im vorher beschriebenen Vektor stehen und das in Abhängigkeit der Variablen k. Diese ist eine lokale Variable, die bei jedem Funktionsaufruf um eins nach oben gezählt wird und beim Erreichen von 21 auf 0 zurück gesetzt wird, damit nicht irgendwelche Elemente des Vektors `output[]` angefordert werden, die es gar nicht gibt. Nun wird in die Spalte ganz links auf der Matrix der Wert des Vektors `output[k]` geschrieben und in die nächste Spalte der Wert von k+1 usw. So sind weiterhin die zusammengehörigen Zahlen zu sehen und da k bei jedem Durchlauf um eins erhöht wird bewegt sich die Schrift mit der Zeit nach links.
Aufgerufen wird die Funktion im Hauptprogramm. Da wir die Matrix schneller laufen lassen wollten als nur mit einer Spalte pro Sekunde haben wir hierfür den Interrupt Vektor des Watchdog-Timers so verändert, dass er zwei Flags ausgibt, die `clock_flag` und die bereits erwähnte `second_flag`. Damit der Watchdog-Timer nun viertelsekündlich einen Interrupt auslöst musste er wie folgt neu konfiguriert werden:

```
  WDTCTL = WDTPW + WDTTMSEL + WDTCNTCL + WDTSSEL + WDTIS0;
  IE1 |= WDTIE;
```

Hier noch einmal die aktualisierte Interruptserviceroutine für den Watchdog-Timer:

```
#pragma vector=WDT_VECTOR
void WDT_ISR()
{ //interrupt vektor fuer time++ sekuendlich und clock++ viertelsekuendlich
  clock++;
  clock_flag |= BIT0;
  if (clock > 3)
  {
    clock = 0;
    time.sec++;
    second_flag = BIT0;
  }
  __low_power_mode_off_on_exit();
}
```

Der Aufruf der Funktion sieht dann folgendermaßen aus:

```
1  if (clock_flag & BIT0)
2 {
3   clock_flag &= ~BIT0;
4   array_output();
5 }
```

Damit ist unser Wecker mit all seinen benötigten und zusätzlichen Funktionen fertig. Er kann die Uhrzeit über das LCD anzeigen, sie lässt sich auch einstellen, es gibt eine Weckfunktion und die LED-Matrix zeigt immer die Aktuelle Zeit an.

Daraus ergibt sich der auf der nächsten Seite folgende Programm-Ablaufplan:

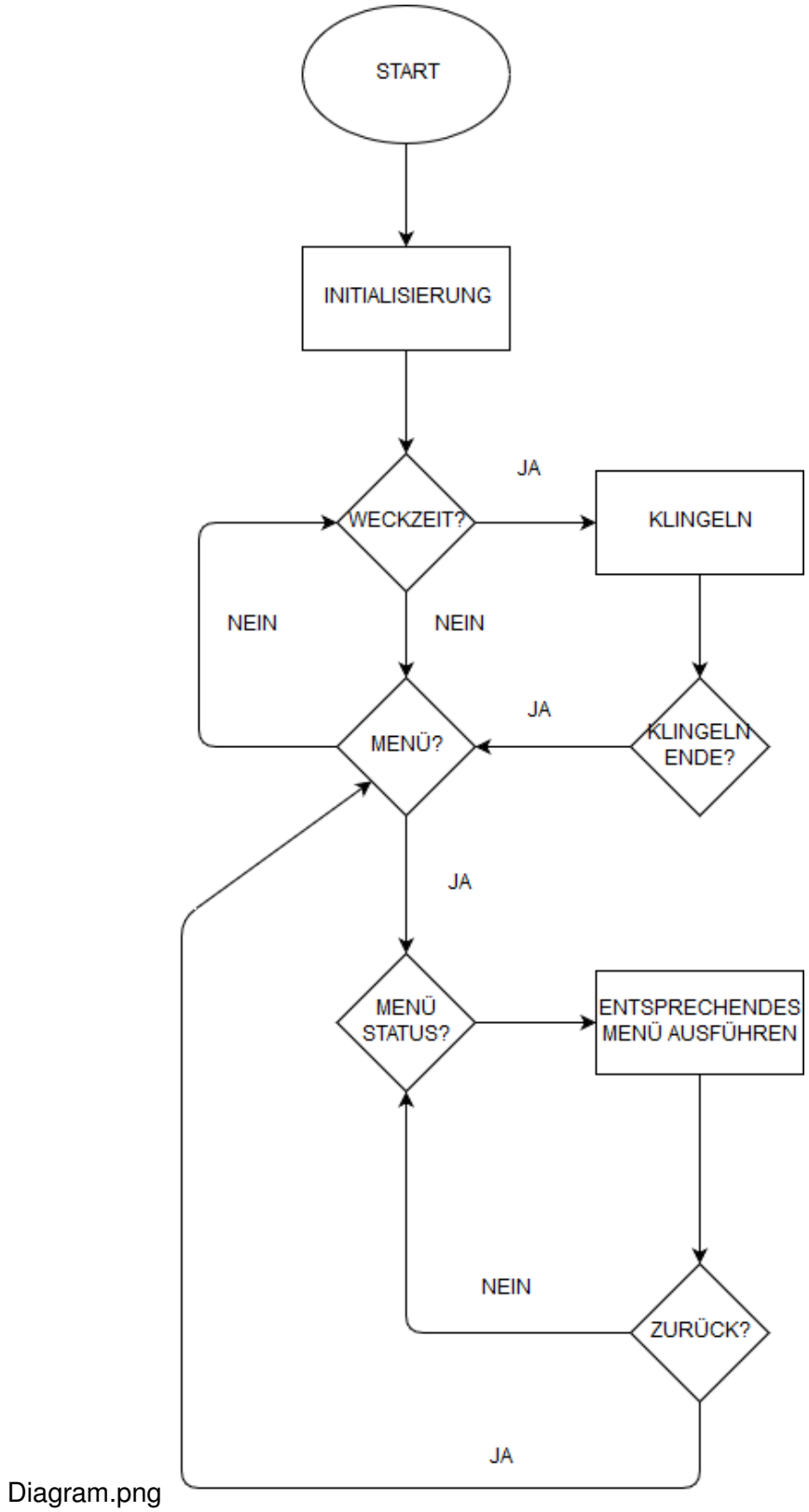
START
INITIALISIERUNG
WECKZEIT?
JA
KLINGELN
NEIN
NEIN
MENÜ?
JA
KLINGELN ENDE?
JA
MENÜ STATUS?
ENTSPRECHENDES MENÜ AUSFÜHREN
NEIN
ZURÜCK?
JA

Diagram.png