|  |  |
| --- | --- |
| **Projekt: DCF77 Wecker** | |
| Standort Wangen/Dübendorf | |
|  | |
|  | |
| Referenz: | **--** |
| Revision: | 0 |
| Version: | 2.0 |
| Ausgabedatum: | <17.12.19> |
| Status: | definitiv |
| Klassifizierung: | öffentlich |
| RUAG Schweiz AG | RUAG Defence  HE\_DU | |



<Projekt DCF77 Wecker>

**Inhalt**

[1 Ausgangslage 3](#_Toc27470102)

[2 Anforderungen des Weckers 3](#_Toc27470103)

[3 Arbeitsschritte 3](#_Toc27470104)

[4 Schema 4](#_Toc27470105)

[5 Nächste Schritte 5](#_Toc27470106)

[5.1 Atmega Fusebit Einstellungen 5](#_Toc27470107)

[5.2 Materialien 5](#_Toc27470108)

[5.3 Steckbrett Aufbau 5](#_Toc27470109)

[5.4 Programm im C 5](#_Toc27470110)

[5.5 Layout Gehäuse 6](#_Toc27470111)

[6 Selbstreflexion 9](#_Toc27470112)

[7 ETC. 9](#_Toc27470113)

[7.1 Überschrift 2 9](#_Toc27470114)

[7.2 Überschrift 2 9](#_Toc27470115)

|  |  |  |
| --- | --- | --- |
| **Genehmigung** | | |
|  | Name | Datum |
| Erstellt: | Alexandra Spring, SPRA | 17.12.19 |
| Geprüft: | Roman Pollak, POLR | 17.12.19 |
| Freigegeben: | Roman Pollak, POLR | 17.12.19 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ausgabenübersicht** | | | |
| Revision | Datum | Beschreibung | Name |
| 1.0 |  | Erstausgabe | SPRA |
| 2.0 | 17.12.19 | Schlussausgabe | SPRA |

# Ausgangslage

Vom Lehrbetrieb wurde mir vorgegeben, dass ich spätestens am 07.45 Uhr im Betrieb erscheinen muss. Da ich aber keinen geeigneten Wecker habe, ist das ein bisschen schwierig. Da ich aber eine auszubildende Elektronikerin bin, dachte ich mir sollte ich in der Lage sein, einen Wecker selber zu programmieren. Ich bin dann in die WDU und habe gefragt ob ich den als nächstes Projekt programmieren darf. Das Projekt wurde bestätigt und ich darf mich nun dem Wecker widmen.

# Anforderungen des Weckers

**Meine persönlichen Anforderungen:**

1. Zeit soll mit einem DCF77 Signal betrieben werden.
2. Falls es zu Unterbrüchen im Signal kommt soll di Zeitanzeige selber weiter zählen können.
3. Die Weckzeit soll beliebig eingegeben werden können.
4. Der Wecker soll temporär ausgeschaltet werden können.
5. Der Wecker soll alle zwei Minuten wieder ertönen.
6. Bei jedem erneuten Wecken soll der Ton aufdringlicher werden.
7. Es soll 4 Tonfolgen haben.
8. Während dem Alarm soll eine Uhr auf dem Display in einer oberen Ecke angezeigt werden.
9. Falls der Wecker aktiviert ist, muss es eine kleine Glocke auf dem Display haben.
10. Es soll ein Countdown auf dem Display angezeigt werden bis zum nächsten Alarm
11. Es soll ein Menü oder eine Rechnung geben die der Schlafende eingeben oder Lösen muss, dass der Wecker definitiv abstellt.
12. Das Ganze soll mit einem Atmega 324P angesteuert werden.
13. Die Hintergrundbeleuchtung des Displays soll während dem Alarm aufleuchten und mittels Taste abstellbar sein.

**Anforderungen von Roman:**

1. Watchdog Timer
2. Alarm in EEPROM einschreiben und beim Falle einer Änderung den Alarm abspeichern und bei Neustart entsprechend setzen.

# Arbeitsschritte

• Schema zeichnen

• Atmega Einstellungen

• Materialien suchen

• Prototyp auf Steckbrett aufbauen

• Programm modifizieren

• Programm erweitern

• Gehäuse Layout

• Gehäuse fertigen

# Schema

# Nächste Schritte

## Atmega Fusebit Einstellungen

Die Fusebit Einstellungen wurden mit der Software Cigwin64 Terminal gemacht. Anfangs war gedacht die Fussebits des Atmega 324p welcher verwendet wird, im AVR Burn – O – Mat zu setzen. Dieses Vorhaben funktionierte leider nicht so wie vorgesehen. Im Burn – O Mat war der Atmega 324 verzeichnet aber die Hintergrundeinstellungen sind nicht korrekt gewesen. Roman hat mir dann geholfen, indem er mir zwei Methoden aufgezeigt hat wie dieses Problem behoben oder umgehen werden kann. Die einfachere und weniger Komplizierte Methode, ist im Terminal die Low Fusebits zu lesen und die High Fusebits zu lesen. Im Datenblatt Kapitel 24 Memory programming unter Fuesbits werden die Bezeichnungen der High Bytes und Low Bytes aufgeführt. Anhand dieser Angaben, können nun die Fusebits neu gesetzt werden. Dies muss aber mit 2 verschiedenen Befehlen gemacht werden. Write hfuses für die High Bytes und write lfuses für die Low Bytes.

## Materialien

Die wichtigsten Materialien hat Roman Pollak mit mitgebracht wie das Display und den Mikrocontroller. Die restlichen mussten aber anderweitig organisiert werden. Die fehlenden Bauteile wurden hauptsächlich in der WDU oder der Lehrwerkstatt rausgesucht.

## Steckbrett Aufbau

Die ersten Steckbrett Aufbauversuche wurden ohne ein Schema aufgebaut. Die Schritte bis zum vollständigen Schema wurden mir von Roman aufgegeben. Der Vorgang bis zum Endresultat war dieser, dass Roman mir eine Aufgabe gegeben hat und ich diese lösen musste. Auf die Art und Weise setze sich der komplette Wecker zusammen. So hatte ich den Überblick, was ich genau machen muss und was ich Neues gelernt habe. Das neu gelernte ging mit dieser Art auch nicht unter und das ganze Projekt war schön aufgebaut mit den entsprechenden Schritten.

## Programm im C

Die Bibliothek welche den DCF77 entziffert wurde aus dem Internet gezogen. Das Programm welche aber den Wecker steuert, habe ich selbst geschrieben mit der Hilfe von Roman Pollak.

Das Programm wird mittels Interrupt Routinen gesteuert. So funktioniert der Wecker zuverlässiger und es können Menupunkte und Einstellungen gemacht werden. Die Zeit wird trotzdem empfangen oder weiter geführt, ohne das die gegenwärtige Aktion diese Funktion unterbricht. Damit ist gewährleistet, dass die Zeit wieder korrekt ausgegeben wird, wenn der Startbildschirm wieder angezeigt wird ohne zu warten, bis das neue Signal ausgewertet wird. Die Zeit wird auch im Hintergrund neu empfangen ohne das die Zeit auf dem Bildschirm gelöscht oder angehalten wird und anschliessend, wenn das Signal korrekt eingelesen wurde wieder ausgibt. Das Anschlussschema und die Bibliothek für das Display habe ich mir aus dem Internet beschafft. Als das Display richtig angeschlossen war, habe ich ein bisschen mit den verschiedenen Funktionen gespielt. Ich habe die Kontraststärke mittels Potentiometer eingestellt und dann angefangen kleine Figuren zu programmieren. Ein Charakter kann maximal 8x5 gross sein. Die Charaktere habe ich mittels Hex Code erstellt. Jedes Bit repräsentiert ein Pixel auf dem LCD – Display. Im Kapitel 5.4.2 gibt es ein Beispiel dazu, wie ein Charakter erzeugt werden kann. Mein erster Schritt für den Wecker war die Zeit ausgeben zu können.Das war ziemlich einfach da in dem heruntergeladenen Bibliothek die Zeit schon ausgewertet wurde und ich dann nur noch die richtigen Funktionen und Variablen finden musste, um die gewünschten Werte auszugeben. Das nächste erste Problem kam aber schon angerast. In der NEMP – Halle gab es keinen Empfang. Die NEMP-Halle ist nichts anderes als ein Faradähischer Käfig. Für diese Erkenntnis ging es eine Weile. Denn ein Signal wurde am Anfang empfangen, aber es war kein DCF77 Signal sondern nur irgend ein Signal welches wilkürlich an den DCF77 Empfänger gesendet wurde. Von wo das Signal kam wusste niemand aber irgendwann haben wir den Versuch mit der richtigen Zeitausgabe im Freien gewagt. Dort entstand aber das Problem, dass der Wind so stark war, dass die Antenne des Empfängers, sich bewegte und diese immer . Durch diese Bewegung empfing die Antenne ein Störimpuls. Die Bibliothek erkannte diesen Störimpuls und wartete wieder auf die nächste Wartesekunde um nochmals das Signal auswerten zu können. Irgendwann sind wir wieder ins Büro und haben gemerkt, dass der DCF77 Empfänger im Gebäude empfängt. An meinem Arbeitsplatz haben wird dann den Empfängern und die Antenne mit verschiedenen Ordner und Bücher beschwert damit diese sich nicht bewegen konnten. Die Drähte die aber vom Empfänger auf das Steckbrett führen, waren nicht gesichert. Aus diesem Grund konnte die Antenne immer noch kein sauberes Signal empfangen. Deshalb war die Idee im Raum, die Drähte zu verlängern und auch diese mit Ordner zu beschweren. Das hat funktioniert und der Empfänger konnte ein sauberes Signal auswerten. Wie wir erkennen konnten, dass kein sauberes Signal empfangen wurde wird in Kaptel 2.2.1 erläutert. Als die Uhrzeit richtig ausgeben konnte und das Display diese sauber angezeigt hat, war der nächste Schritt ein Menu zu programmieren. Dort hatte ich ziemlich lange, bis ich eine Lösung gefunden habe. Mit einer Switsch -Schleife habe ich eine State Machine erstellt. Essentiell eine Switschschleife welche nur mit eindeutigen, von Nutzern definierten Befehlen, kontrolliert werden kann. So konnte ich kontrollieren in welche Zustände gewechselt werden können im aktuellen Status. So konnte ich auch einen Zurück Befehl einbauen welcher zuverlässig funktioniert und die gleichen Zustände an mehreren Stellen verlinken. Diese Lösung funktioniert ziemlich gut und bereitet keine Probleme. Nur die Lösungsfindung hat lange gedauert. Als nächstes war dann die Weckzeit einstellung an der Reihe. Dort können nur die Minuten und Stunden eingestellt werden. Denn die Sekundeneinstellung ist in diesem Fall völlig überflüssig. Für Minuten und Stunden gab es je zwei Variablen, also insgesamt gibt es vier einzustellende Ziffern. Alle Ziffern können unabhängig von einander eingestellt werden, sodass nich 40 mal eine Taste gedrückt werden muss um 40 Minuten einstellen zu können. Sondern nur eine Vier und eine Null. Natürlich gibt es noch einen Taster um zwischen den Ziffern wechseln zu können. Die Ziffern können beliebig lange eingestellt werden, bis der bestätigungs Taster gedrückt wird. Es wird dann die eingestellte Weckzeit angezeigt und gleichzeitg werden die Variabeln der Minute zusammen gezählt und die Variabeln der Stunde. So können die Weckzeit und die aktuelle Zeit vergliechen werden. Es muss nochmals ein Taster gedrückt werden zur zweiten Bestätigung, welcher direkt zum Hauptbildschirm zurück führt. Jetzt wird aber nicht automatisch der Alarm läuten zu gewünschten Zeit, wenn nicht die nötigen Massnahmen getroffen wurden. Diese Massnahme, ist auch der Grund welcher den Aufzuweckenden dazu bringt aufzuwachen. Denn der Wecker muss zuerst aktiviert werden. Das wird alles im Menu gemacht. Wenn der Alarm aktiviert ist und es Zeit ist aufzuwachen, dann geht der Alarm los. Um den Alarm zu erstummen zu bringen kann die Snooze Taste gedrückt werden. Der Alarm wird aber dadurch nur temporär ausgeschaltet. Nach drei Minuten erklingt er wieder von neuem bis auch der wieder zu erstummen gebracht wird. Der nächste Alarm kommt aber wieder nach 3 Minuten. Das geht immer weiter so, bis der Wecker deaktiviert wird. Das wird auch mit dem Menu gemacht. Nach jedem temporären deaktivieren des Alarms wird wieder der Hauptbildschirm angezeigt. Von dort aus, kann alles normal bedient werden, bis der nächste Alarm erklingt. Über das Menu kann nun der Wecker definitiv deaktiviert werden. Der Wecker muss wieder aktiviert werden, wenn die Person will, dass der Alarm wieder an der gleichen Zeit losgeht. Die Zeit bleibt gespeichert und jetzt muss der Wecker nur noch wissen ob er ertönen soll. Wie die Zeit und der Alarm gespeichert werden siehe bitte Kapittel 2.3.3. Der Alarmton wird mit verschiedenen Impulsfolgen gesteuert. So kann der Alarmton aufdringlicher werden und der Aufzustehenden auf die Dringlichkeit des Aufwachens hinweisen. Der Alarm wird wiederholt abgespielt bis dieser definitiv deaktiviert wird. Der Weckbefehl, ist dann nicht mehr vorhanden. Der Weckbefehl, wird erst wieder aktiv, wenn er im Menu wieder aktiviert wird. Die Zeit bleibt unverändert, bis solche geändert wird. Es können nicht mehrere Zeiten abgespeichert werden. Das heisst, wenn eine andere Weckzeit gewünscht ist, muss diese wieder manuel eingegeben werden und die neue Weckzeit wird abgespeichert. Die alte Weckzeit wird dadurch gelöscht und kann nicht mehr abgerufen werden.

Die Gesamtstunden und GesamtMinuten werden separat abgespeichert und ausgegeben. Die Daten werden in ein EEPROM abgespeichert. Auf diese Art und Weise wird sicher gestellt, dass bei einem möglichen Stromfall über Nacht, welcher unbemerkt bleibt, am nächsten Morgen die Person trotzdem geweckt wird. Im EEPROM wird die Weckzeit und die Information ob der Alarm gesezt ist. So werden diese Daten nicht jedes mal eingestellt werden müssen, wenn aus irgend einem Grund der Wecker neu gestartet wird. Der nächste Schritt war den Alarmton zu programmieren. Zuerst habe ich ihn so programmiert, das bei jedem drücken des Snooze Tasters eine Variabel hinaufgezählt wird. Der Alarm wird mit jedem drücken um eine vordefinierte Taktzahl höher. Aber damit auch das Aufstehen sofort garantiert wird, musste der Alarmton sogar noch aufdringlicher gestaltet werde. Es gibt fünf verschiedenen Taktgeschwindigkeiten. Beim ersten erklingen des Alarmes wird die langsamste Impulsfolge ausgegeben nach zehn Sekunden wird die nächst schnellere Impulsfolge abgespielt. So geht das weiter, bis die aufdringlichste Impulsfolge von allen ertönt. Es gibt aber sogar noch eine zusätzliche Funktion. Mit jedem erneuten erklingen des Alarm wird die am wenigsten aufdringliche Alarmstufe ingoriert. Nach vier Mal drücken des Snooze Tasters heisst das also, dass nur noch die aufdringlichste Alarmfolge ertönt und somit der Aufzustehende Weis, das er sofort aufstehen muss, oder er kommt zu spat. Dort war die schwierigkeit den Code so klein wie möglich zu behalten ohne das irgend ein Feature ausgelassen wird. Ich habe das mit einer Berechnung und einem Switch schleife gelöst.

Das Programm welche den DCF77 entziffert wurde aus dem Internet gezogen. Das Programm wird mit Interrupts gesteuert. So kann programmiert werden ohne dass der Mikrocontroller zu lange an einem Delay festhalten, wenn er in dieser Zeit etwas anderes erfüllen könnte. Das Programm welche aber den Wecker steuert, habe ich selbst geschrieben mit der Hilfe von Roman Pollak. Die Informationen für die Display Ansteuerung habe ich aus dem Internet geholt und zuerst mit den verschiedenen Funktionen ein bisschen herumgespielt um mein Feeling dafür zu verstärken und zu festigen. Als nächsten wurde wirklich der Wecker selber programmiert. Die Zeit Ausgabe und Datum war kein Problem aber die Menu Programmierung war schon ein anderes Thema, das habe ich mit States gelöst. Das war das schwierigste an dem Ganzen. Das Menu, alle Zeichen und Funktionen beachten welche im aktuellen Menupunkt auszuführen sind. Die restlichen Punkte hingegen waren im Vergleich zur Menu- programmierung schon fast einfach und gingen ziemlich schnell vorwärts. An der Edgetriggerung hatte ich auch eine Weile. Denn bis jetzt habe ich immer die Flanken des ganzes Ports eingelesen aber in der jetzigen Lösung wird jeder einzelne Pin nach der Flanke abgefragt ohne das es zu viele zusätzliche Codezeilen gibt.

### C Programm

### Charakter erstellung

### USART

### EEPROM

## Layout Gehäuse

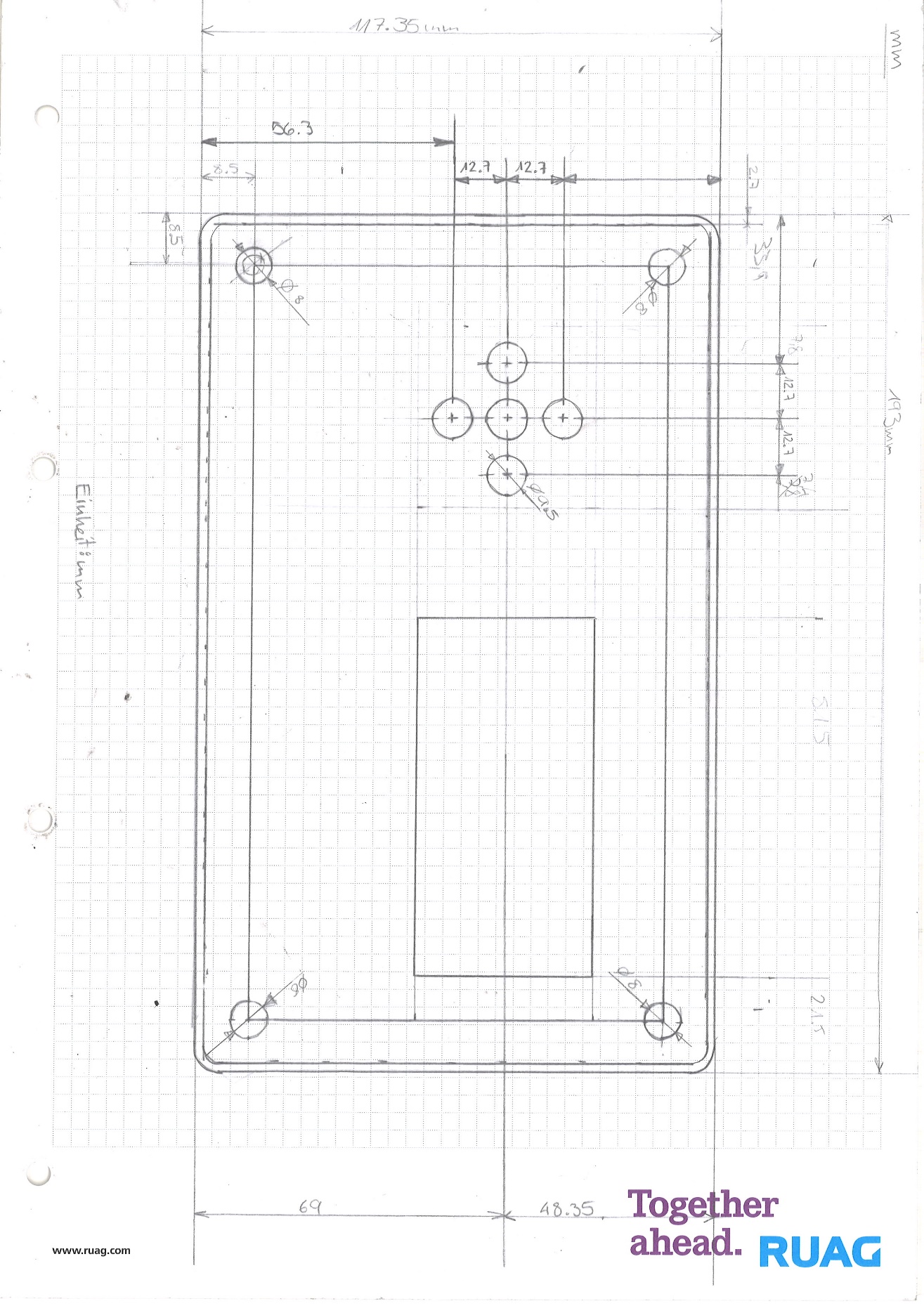
Die Teile des Weckers wurden ausgemessen und die Massen aufgeschrieben. Danach habe ich eine Skizze gemacht des Gehäuses ohne eingeschriebene Masse. Ich habe die Mindestmasse der ausgemessenen Teile zusammengerechnet und noch die mindest Abstände zwischen den Teilen und den Gehäusen dazugerechnet. Damit habe ich jetzt die mindestmasse eines Gehäuses welche ich brauche um überhaupt ein richtiges Gehäuse aszusuchen. Dieses Gehäuse muss aber natürlich noch weiterverarbeitet werden. Die Schalter müssen aus dem Gehäuse geführt werden, sowie auch das Netzkabel und den DCF77 Empfänger. Natürlich muss das Display sichtbar sein damit ich die Zeiten und Angaben lesen kann. Für diese Teile musste ich Aussparungen bohren und Feilen.

### Mechanische Arbeiten

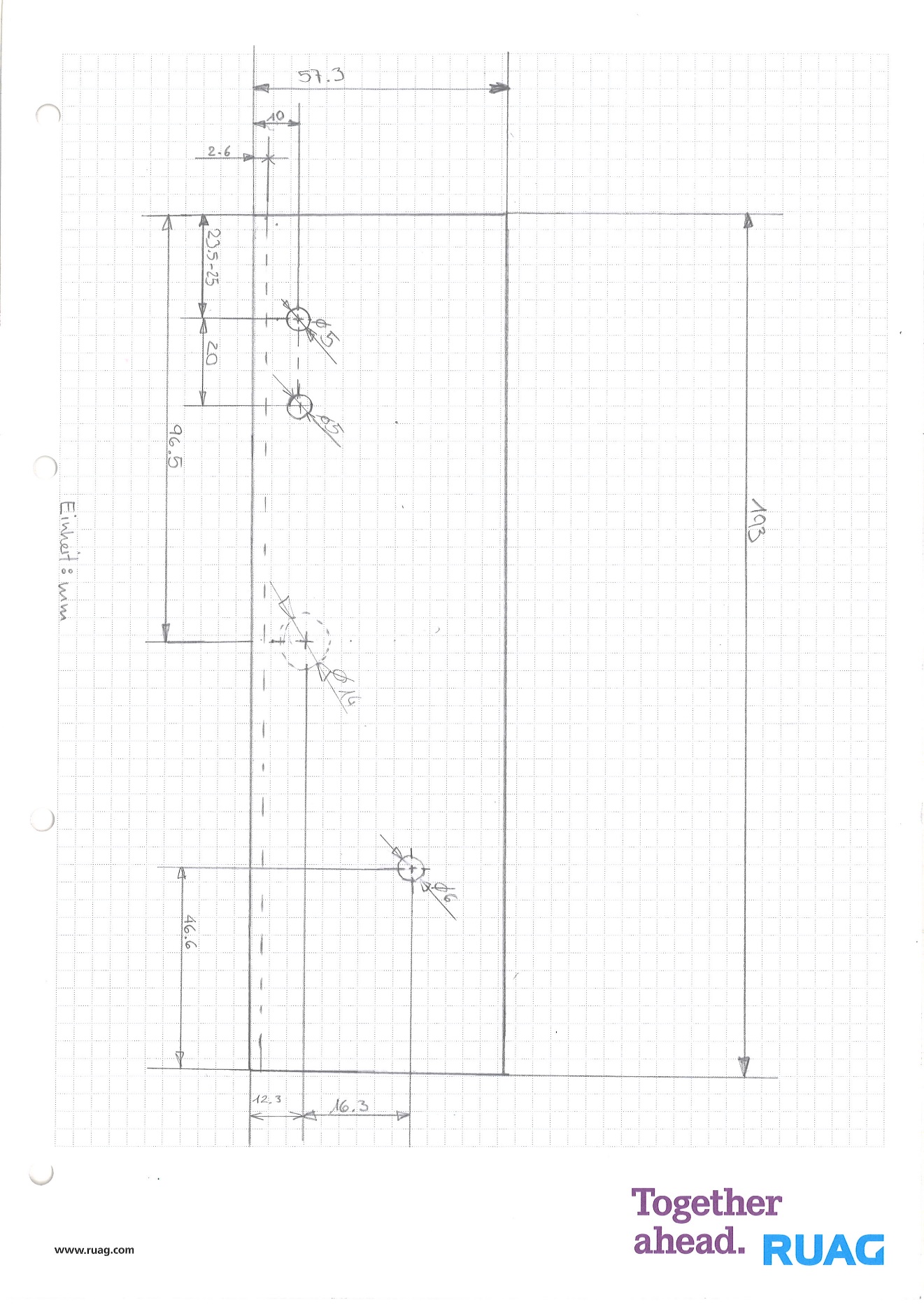
* Menutaster Öffnung
* Alarmtaster Öffnung
* Display aussparung
* Summeröffnung
* Netzkabel ausführung
* DCF77 Empfänger ausführung

### Mechanische Skizzen

#### Ansicht Front



#### Ansicht Unten



# Selbstreflexion

Bei der Menuprogrammierungen habe ich extrem lange gebraucht und am Schluss doch nicht wirklich einen Anhaltspunkt gehabt. Irgendwann habe ich Roman gebeten mir zu helfen. Ich brauchte nur einen Anhaltspunkt um weiter zu kommen und von da an ging es ziemlich einfach. Hätte ich früher nachgefragt, dann währe das Problem schneller gelöst gewesen und ich hätte mit dem nächsten Schritt anfangen können.

Ich habe Probleme mit den Register Descriptions im Datenblatt. Ich brauche sehr lange um diese zu lesen und zu setzen. Ich konnte das Datenblatt nicht richtig interpretieren uns somit waren die Register Einstellungen nicht komplet korrekt. Jedes Mal habe ich an den selbst interpretierten Einstellungen gezweifelt

Ich muss unbedingt von Anfang an fragen was ich nicht verstanden habe an der Aufgabenstellung und mir ein bisschen mehr zutrauen. Ich sollte auch ein bisschen mehr ausprobieren und nicht nur überlegen und Fragen und eine Lösung haben. Ich sollte auch Raum für Fehlschläge lassen und nicht nur für Erfolge, welche ich schlussendlich vielleicht nicht einmal erreiche.

# ETC.

In dieser Woche habe ich gelernt wie man ein Menu im C programmiert. In meinem Fall habe ich das mit einem Switch case gemacht, aber habe mir noch eine andere Methodik erklären lassen. Ich hätte früher nach einem Anhaltspunkt fragen können. Ich weiss jetzt wie man mit einem Compiler die Fusebits eines Mikrocontrollers lesen und neu setzen kann. Vorher habe ich das mit dem AVR Burn O Mat gemacht aber das Programm konnte den Atmega 324 nicht lesen, weil der im File nicht als solcher angegeben war. Deshalb hat Roman mir gezeigt wie das anders gemacht werden kann und wo man das im Datenblatt auslesen kann und nicht auf engbedded.com. Ich habe gelernt wie man mit Pointern arbeiten kann und wie die funktionieren. Ich hätte von Anfang an beim Subprogramm Fragen sollen und bei diesem Punkt genau mein Programm durchlesen und meine Flüchtigkeitsfehler mit der Umstrukturierung selber bemerken können.

Ich habe Probleme mit den Register Descriptions im Datenblatt. Ich brauche sehr lange um diese zu lesen und zu setzen und am Schluss sind sie nicht komplett richtig und ich bin mir selber unsicher. Ich muss unbedingt von Anfang an fragen was ich nicht verstanden habe an der Aufgabenstellung und mir ein bisschen mehr zutrauen. Ich sollte auch ein bisschen mehr ausprobieren und nicht nur überlegen und Fragen und eine Lösung haben. Ich sollte auch Raum für Fehlschläge lassen und nicht nur für Erfolge, welche ich schlussendlich vielleicht nicht einmal erreiche.

## Überschrift 2

## Überschrift 2

### Überschrift 3