Track your Cat – GPS-Empfänger
Projekt

Abschlussbericht

im Studiengang
Technische Physik

vorgelegt von

Patrick Gröppner (Matr.-Nr. 00750805)
Thomas Hofmann (Matr.-Nr. 00740490)

am 8. Februar 2021
an der Technischen Hochschule Deggendorf

Prüfer: Prof. Dr. Josef Kölbl

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das GPS ist aus unserem alltäglichen Leben nicht mehr wegzudenken. Das beste Beispiel hierfür ist sicherlich das Navigationssystem im Auto. Um aber die GPS-Ortung nutzen zu können benötigt man dafür ein passendes Empfangsgerät. Es gibt schon seit längerer Zeit sogenannte „GPS-Tracker“ auf dem Markt, welche eine SIM-Karte benutzen zur Datenübertragung. Die Nutzung solch eines Gerätes verlangt jedoch dann in Deutschland einen monatlich laufenden Mobilfunkvertrag.

1.1 Problem- und Zielstellung

Das Ziel dieses Projektes ist deshalb einen kleinen und handlichen GPS-Empfänger zu entwickeln, der keine laufenden Betriebskosten besitzt und leicht in der Bedienung und Auslesung ist. Das Gerät soll vornehmlich dafür gedacht sein, um Haustiere verfolgen zu können oder deren Tagesablauf darzustellen. Darüber hinaus ist zuerst die Konzeption eines Prototypens angepeilt.

Zudem ist eine maximale Akkulaufzeit angestrebt, sodass der Empfänger mindestens eine Woche durchgehend benutzt werden kann und die ermittelten Positionen festhält.

Außerdem soll ein ausschlaggebendes Datenblatt erstellt werden, welches alle Vorteile und Eigenschaften des neuen Geräts auflistet.

1.2 Eingeschlagener Lösungsweg

Kurz nach dem Beginn des Projekts kam man zu dem Entschluss, dass das durchgehende Senden der aktuellen Position des Haustieres über eine Mobilfunkverbindung nicht nur anhaltende monatliche Kosten darstellt, sondern auch die Akkunutzung erhöht.

Deshalb wurde sich dazu entschlossen den Empfänger als Datenschreiber zu konzipieren. Die von einem GPS-Chip einhergehenden Positionsdaten sollen durchgehend Zeile für Zeile auf eine eingesteckte Micro SD-Speicherkarte geschrieben werden.

1.3 Durchführung und Zusammenarbeit

Damit das Projekt ein Erfolg werden kann, musste die Arbeit untereinander aufgeteilt werden. Zu den Hauptaufgaben zählen der Aufbau der Hardware und das Programmieren des Mikrokontrollers, welcher sozusagen den „Kopf“ des Empfängers darstellt. Die auszuführenden Tätigkeiten der Projektarbeit teilten sich folgendermaßen auf:

- Optimierung und Schreiben des Programmcodes als C-Programm mithilfe der Arduino Entwicklungsumgebung
- Entwicklung der Empfangsplatine:
    - Schaltplan
    - Boardlayout
- Kleinere Aufgaben erledigen wie Internet-Recherche oder Planung der nächsten Treffen
- Dokumentation der Projekttätigkeiten
- Suchen nach passenden Teilkomponenten für das Board Design
- Konstruktion eines passenden Gehäuses (hinsichtlich Radioempfang und Schutz gegen äußere Umweltbedingungen)
- Erstellung eines Produktdatenblattes

2 Systementwurf

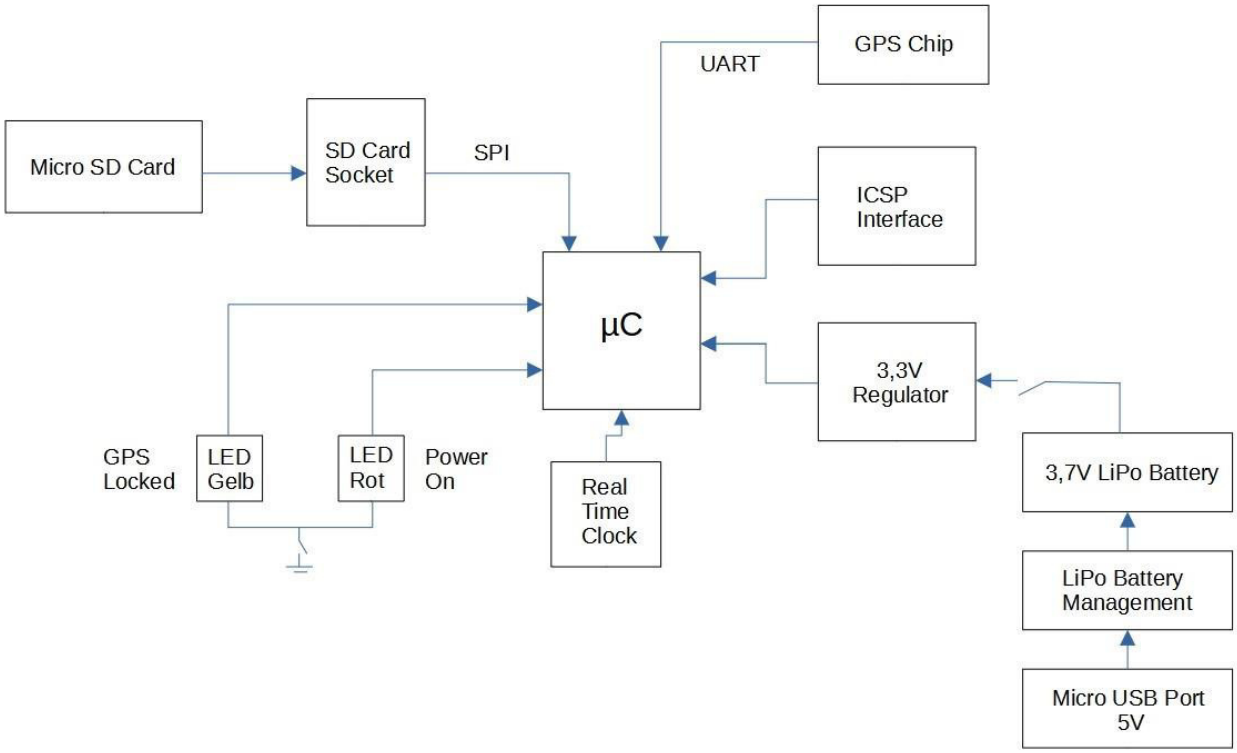


Abb. 1: Blockdiagramm des Empfängers, Quelle: Eigene Darstellung

Nun musste man sich überlegen, wie man solch ein Gerät konzipieren könnte. In Abbildung 1 ist das endgültige Blockschaltbild des Empfängers dargestellt. Jedoch soll nun aufgezeigt werden, warum einzelne Komponenten oder warum gerade diese Anordnung gewählt wurden.

Als zentrale Koordinierungseinheit des Empfängers musste zuerst ein passender Mikrokontroller ausgewählt werden. Zwecks einer erleichterten Programmierung und Zugriff auf viele benötigte Bibliotheken der Arduino-Entwicklungsumgebung fiel die Wahl früh auf einen 8-bit Mikrokontroller der Firma Atmel. Diese sind weit geläufig und haben einen geringen Kostenfaktor. Ausgewählt wurde einen ATMEGA 328P, denn dieser besitzt 32kByte Speicher – genug für spätere Softwareerweiterungen oder ähnliches. Weitere Auswahlkriterien für diesen Chip waren ausreichend viele Schnittstellen wie SPI, I²C oder UART, eine Arbeitsspannung von 3,3 Volt und eine Taktfrequenz von 8 MHz. Die UART Verbindung sollte auch softwareseitig realisiert werden können.

Die Positionsdaten sollen auf eine SD-Karte geschrieben werden. Dafür ist eine Einsteckbuchse für diese Karte nötig gewesen. Durch den 3,3 Volt Pegel ist kein Logikpegel-Wandler nötig. An diese Positionsdaten gelangt das Gerät, indem der Mikrokontroller einen GPS-Empfangschip über die UART Schnittstelle ansteuert und ausliest. Ausgewählt wurde ein ORG1410 von OriginGPS. Jener besitzt mehrere Vorteile gegenüber anderen Empfangschips auf dem Markt. Zum einen ist der Chip mit einer fest

integrierten Patch-Antenne ausgestattet, welche im Gehäuse vor äußeren Wettereinflüssen geschützt ist. Des Weiteren hat der Chip selbst während der aktiven Suche nach GPS-Signalen einen Stromverbrauch von maximal 50 mA. Darüber hinaus war er schon im Lagerbestand der Hochschule vorhanden. Somit konnte man einige Kosten für den Prototypen einsparen.

Zudem soll der Empfänger über eine wiederaufladbare Batterie betrieben werden. Darum werden andere Komponenten zum reibungslosen Laden benötigt. Gewählt wurde der BQ24040 als Lade-IC für die Batterie. Dieser zeichnet sich durch seine sehr geringe Stromaufnahme und eine Betriebsspannung von 5V aus. Letztere ist durch eine einfache Versorgung über ein Handy-Ladekabel oder zweitrangig den USB-Port eines Computers ausgezeichnet geeignet. Auch dieser war er schon im Lagerbestand der Hochschule vorhanden.

Das bestimmt wichtigste Kriterium war aber eine geringe Stromaufnahme des Geräts, um so eine Batterielaufzeit von mindestens einer Woche gewährleisten zu können. Jedoch kann die Stromaufnahme der SD-Karte beim Schreiben und Auslesen des GPS-Chips zusammen mit der des Mikrokontrollers kurzzeitige Spitzenwerte von bis zu 300mA erreichen. Dies bedeutet zweierlei Dinge:

1. Für den 3,3 Volt Pegel ist ein Spannungsregler von Nöten, der mit einem zusätzlichen Sicherheitspuffer von 100 mA diese hohe Stromaufnahme durchgehend liefern könnte. Deshalb wurde der TPS74533 von Texas Instruments ausgewählt. Dieser zeichnet sich durch seine enorm niedrige Aussetzspannung von weniger als 200mV über der 3,3 Volt Nennspannung bei 500mA Stromdurchsatz und sehr kleinem Formfaktor aus.
2. Für die lange Batterielaufzeit muss der „Deep Sleep“ Modus des Mikrokontrollers über einen Zeitraum von mindestens 20 Minuten gefahren werden, damit sich die mittlere Stromaufnahme stark verringert. Dieser lange Zeitabstand erfordert einen präzisen Zeitgeber. Die Wahl fiel auf den MCP7940M von Microchip Technology, der sich durch seine niedrige Stromaufnahme von 1,2µA, eine I²C-Ansteuerung und bereits existierende Programmbibliotheken auszeichnet.

2.1 Bedienung des Empfängers

Darüber hinaus soll nun noch darauf eingegangen werden, wie genau der Empfänger bedient werden soll. Gestartet wird das Gerät, indem man einen Stromschalter betätigt. Nach einigen Stunden oder Tagen, wenn die Katze wieder zuhause ist, kann die Verfolgung durch das erneute Betätigen des Stromschalters beendet werden. Die Gerätschaft ist sodann ausgeschaltet und die SD-Karte kann entnommen werden.

Am Ende der Überwachung soll der Nutzer daraufhin diese Karte in einen Computer oder Laptop stecken. Mit der kostenlosen Software „Google Earth Pro“ kann man sich nun die einzelnen zeitlich nacheinander folgenden Punkte auf der Weltkarte anzeigen lassen. Das Dateiformat der empfangenen Daten ist das NMEA Format, welches direkt in Google Earth importiert werden kann.

3 Die Entwicklung eines Schaltplanes

Nachdem ein Blockdiagramm zur Funktionsweise des Geräts erstellt und die wichtigsten Teilkomponenten ausgewählt wurden, sollte nun ein Schaltplan erstellt werden. Dieser wurde mithilfe der öffentlich zugänglichen Entwicklungssoftware „KiCad“ der Version 5.1.9 erstellt.

3.1 Die Spannungsversorgungen

In diesem Projekt wurden lineare Spannungsregler gegenüber Schaltreglern bevorzugt wegen folgenden Aspekten: Diese benötigen weniger Platz und erzeugen viel weniger Störsignale, die eventuell auf die Datenleitungen einkoppeln könnten. Zudem wird sich das Gerät die meiste Zeit über im Tiefschlafmodus befinden, in dem es nur einige µA an Strom benötigt. Bei so niedrigen Strömen sind lineare Spannungsregler immer noch effizient genug für dieses Projekt.

3.1.1 3,3 Volt Spannungsversorgung

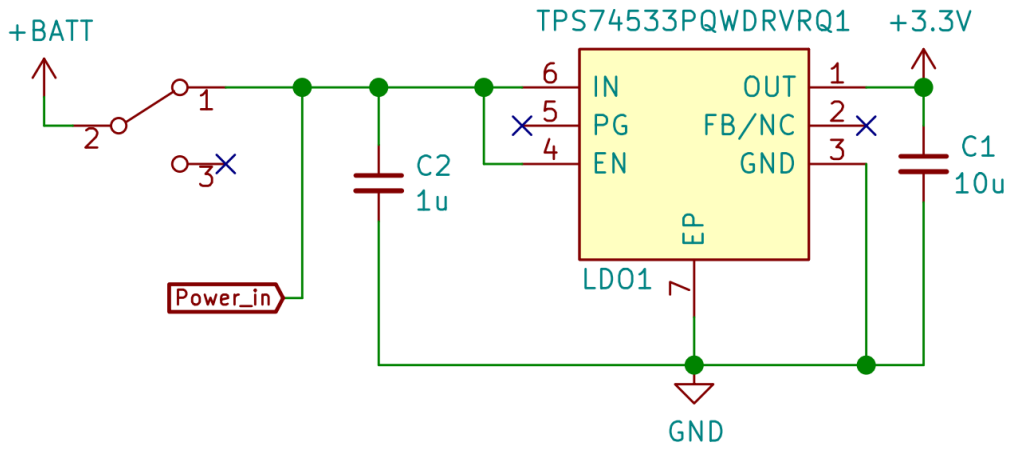


Abb. 2: Beschaltung des TPS74533, Quelle: Eigene Darstellung

Abbildung 2 zeigt die grundlegende Beschaltung des TPS74533, dem Spannungsreger für den 3,3 Volt Pegel. Es wurde ein einfacher Schalter verwendet, um die Spannungszufuhr aktivieren bzw. deaktivieren zu können.

C1 und **C2** sind Entkoppelkondensatoren, die die Ein- und Ausgangsspannung glätten und als Tiefpassfilter für hochfrequente Störungen wirken sollen. Laut Hersteller sollte C1 mindestens 1uF an Kapazität besitzen, um die Stabilität des Reglers nicht zu gefährden. [1, S. 4]

Da die Batteriespannung nie unter 3,3 Volt fallen kann, ist der „Enable“-Pin Nr. 4 auf diesen Pegel hochgezogen. Das bewirkt einen durchgehenden Betrieb des Reglers.

3.1.2 1,8 Volt Spannungsversorgung

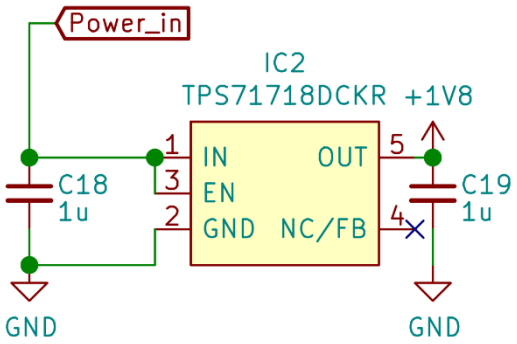


Abb. 3: Beschaltung des TPS71718 Reglers, Quelle: Eigene Darstellung

Abbildung 3 zeigt die grundlegende Beschaltung des TPS71718, dem Spannungsreger für den 1,8 Volt Pegel des GPS-Empfangschips. Auch hier regelt der gleiche einfacher Schalter die Spannungszufuhr.

C18 und **C19** sind Entkoppelkondensatoren, die die Ein- und Ausgangsspannung glätten und als Tiefpassfilter für hochfrequente Störungen wirken sollen.

Da die Batteriespannung nie unter 3,3 Volt fallen kann, ist der „Enable“-Pin Nr. 4 auf diesen Pegel hochgezogen. Das bewirkt einen durchgehenden Betrieb des Reglers.

3.2 Der Mikrokontroller

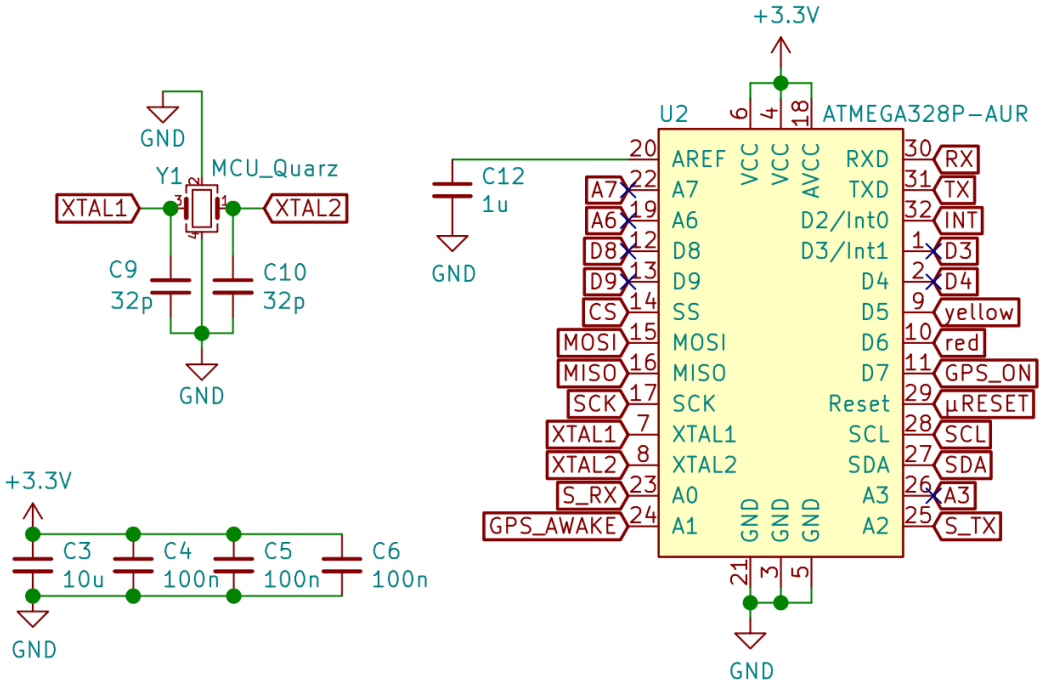


Abb. 4: Die Beschaltung des Mikrokontrollers, Quelle: Eigene Darstellung

Abbildung 4 zeigt die Beschaltung des **ATMEGA328P**, dem zentralen Mikrokontroller des Systems. Damit die Design-Software für die einzelnen Pins (und deren Verbindungsnetze auf dem Board später) keine zufälligen Zahlen verwendet, wurden alle Pins mit aussagekräftigen globalen Labels (engl. Bezeichner) versehen. Das Symbol ist

selbsterstellt, da die Arduino-IDE eigene Pinbezeichnungen und nicht die des Herstellers verwendet.

C3 bis **C6** sind die Entkoppelkondensatoren für die Spannungsversorgungs-Pins 4, 6 und 18. Der Hersteller empfiehlt so nahe wie möglich an diese Pins einzelne Keramik-Kondensatoren mit 100nF Kapazität zu platzieren. Der Kondensator C3 ist mit 10µF um 100 Größenordnungen größer und dient als sehr schneller „Zwischenspeicher“ für die kleineren Kondensatoren.

C12 ist der Referenzkondensator des Analog-Digital-Wandlers. Zwar wird diese Funktion von Gerät nicht genutzt, aber Atmel empfiehlt dieses Bauteil trotzdem zu verwenden.

Y1 ist der externe 8MHz-Quarz-Resonator, ohne den der Mikrokontroller nicht zuverlässig arbeiten könnte. Der interne RC-Oszillator des Chips ist zu ungenau. Aus Platzgründen wurde eine Variante mit 4 Pins gegenüber einem standardmäßigen Resonator mit 2 Pins vorgezogen.

C9 und **C10** sind die beiden Ziehkondensatoren des Resonators Y1. Sie müssen beide den gleichen Wert haben. Aus Sicht des Schwingquarzes sind diese beiden Kondensatoren in Serie geschalten. Die Lastkapazität des Resonators kann über

$$C_{Last} = \frac{C_1 \cdot C_2}{C_1 + C_2} + C_{Streu} \qquad [6]$$

berechnet werden. Damit ergeben sich für die beiden Ziehkondensatoren die Werte

$$C_{1,2} = 2 \cdot C_{Last} - C_{Streu} \; .$$

Jedoch wirkt die gesamte Leiterplatte selbst auch wie ein paralleler Kondensator auf Y1 ein. Somit muss man immer noch eine Streukapazität einplanen. Ein Wert von 3-6 pF ist eine gute Schätzung.

3.3 Der Empfangschip

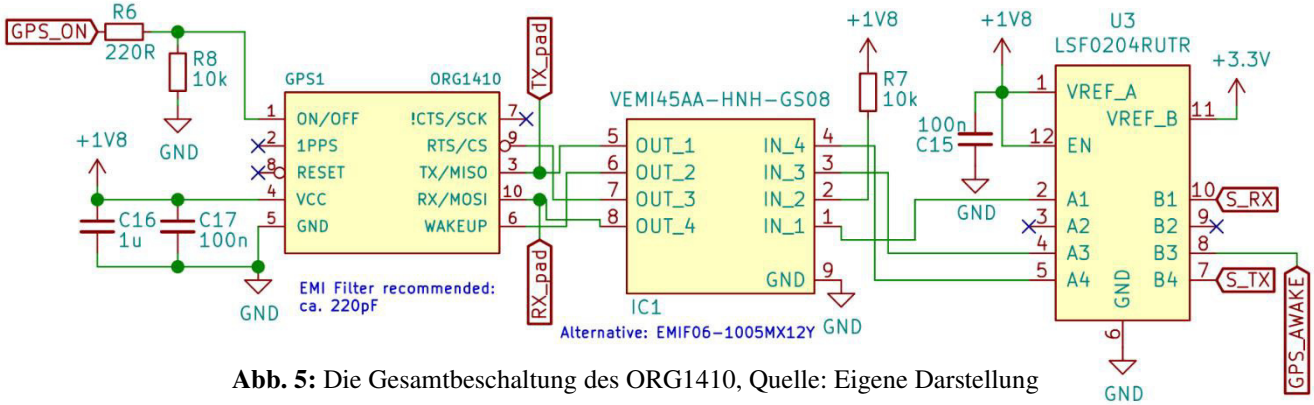


Abb. 5: Die Gesamtbeschaltung des ORG1410, Quelle: Eigene Darstellung

Abbildung 5 zeigt die Beschaltung des **ORG1410** von OriginGPS, dem integrierten Schaltkreis zum Empfangen von GPS-Nachrichten und der Bestimmung des aktuellen Standortes. Die Pins 2, 7 und 8 werden nicht benutzt und müssen deshalb in der Design Software mit einem „X“ markiert werden. Dies sorgt dafür, dass das Netz dieser einzelnen Pins isoliert wird.

C16 und **C17** sind die Entkoppelkondensatoren des ICs und sind für den störungsfreien Betrieb nötig. **C17** muss so nahe wie möglich am Versorgungspin platziert sein und hat deswegen einen Wert von 100nF. [1, S. 26]

R8 ist ein Pulldown-Widerstand, der den Pin 1 auf Masse zieht und vom Hersteller mit 10kΩ empfohlen wird. Auf diesen Pin gibt der Mikrokontroller einen kurzen Spannungsimpuls (vgl. Kapitel 4) aus, um den Chip zum Hoch- oder Herunterfahren zu bewegen. [1, S. 24]

Zudem empfiehlt OrginGPS die Datenleitungen durch ein EMI Filter zu verlegen (hier **IC1**), welches zwei Kapazitäten von etwa 220pF hat. Das ausgewählte Filter besitzt eine Pi-Struktur und wirkt als Tiefpass gegen sehr hochfrequente Störungen (f_{3dB} = 100 MHz).

R7 ist ein Pullup-Widerstand, der den Pin 9 über das Filter auf die Versorgungsspannung des ICs zieht und vom Hersteller mit 10kΩ angegeben ist. Dieser Widerstand sorgt dafür, dass der Empfangschip beim Hochfahren als Bussystem eine UART-Verbindung mit dem Mikrokontroller aufbaut. [1, S. 24]

Die globalen Bezeichner „**RX_pad**“ und „**TX_pad**“ sind dafür da, um in einer späteren Testphase mit einem Logik Analysator die RX und TX Datenleitungen extern messen und auslesen zu können.

Der IC mit der Bezeichnung **U3** ist ein Spannungspegel-Wandler, damit der 3,3 Volt Mikrokontroller mit dem 1,8 Volt Empfangschip kommunizieren kann. Davon sind die UART Datenleitungen und die Ausgänge des ORG140 betroffen. Die Eingangspins dieses Chips sind 3,3 Volt tolerabel und sind deshalb direkt verbunden (vgl. Pin 1 in Abb. 5).

C15 ist ein Entkoppelkondensator für den Pegelwandler und dient als dessen Energiespeicher für den 1.8 Volt Pegel. Für den anderen Pegel ist laut Hersteller kein Kondensator zwingend nötig (dazu in Kapitel 5.3 mehr).

4 Der Ablauf des Programmcodes

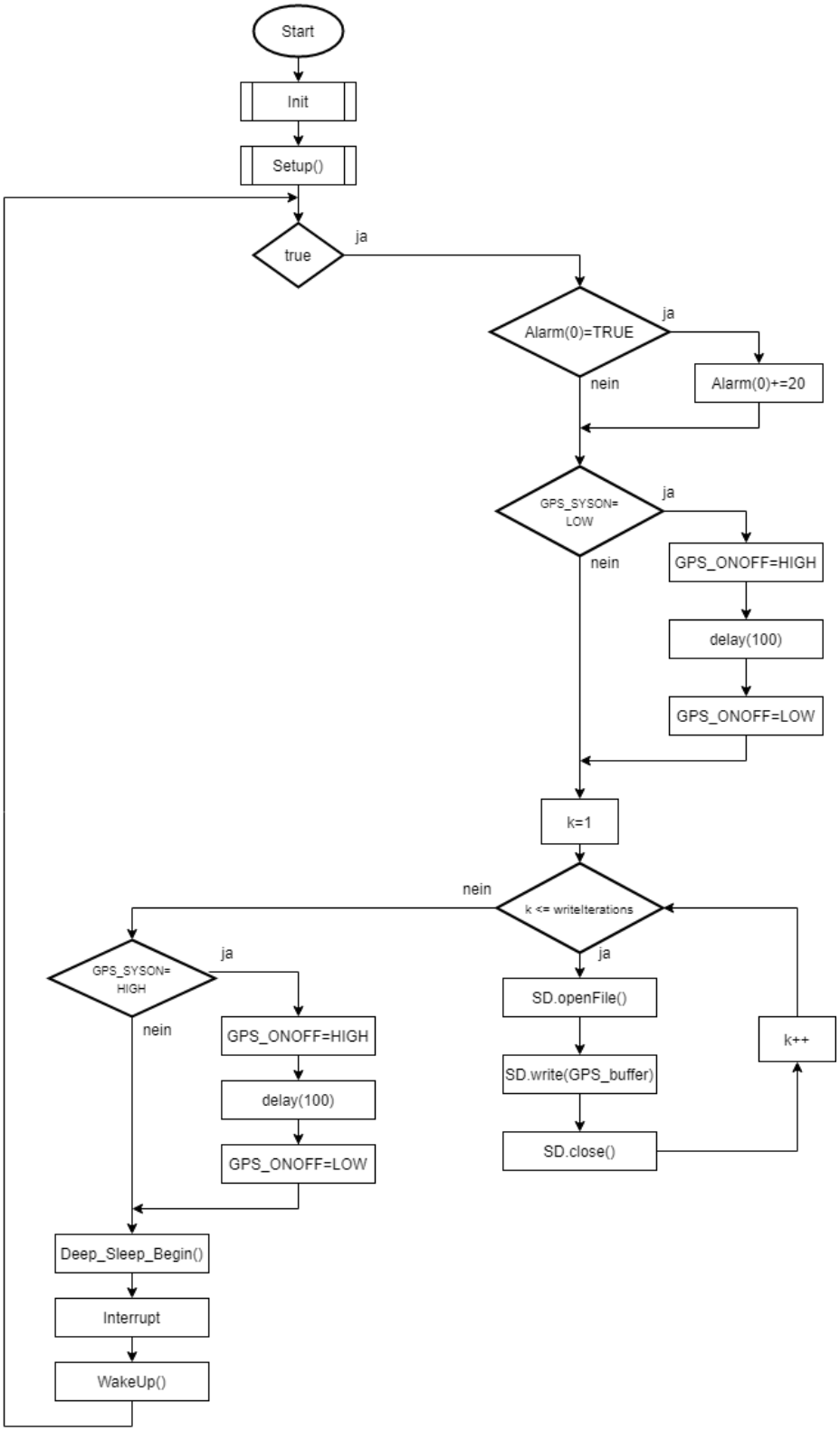


Abb. 6: Flussdiagramm des Programmcodes, Quelle: Eigene Darstellung

Nachdem der erste vollständige Schaltplan erstellt wurde, konnte mit der Programmierung des Mikrokontrollers begonnen werden. Dies geschah mithilfe der integrierten Arduino Entwicklungsumgebung (IDE). Um den Programmcode und Ablauf der einzelnen Sequenzen nun besser darstellen zu können wurde ein Flussdiagram erstellt (siehe Abb. 6). Jede Raute steht dabei für eine if-Anweisung.

Nach dem Einschalten der Spannungszufuhr und dem Booten (engl. Hochfahren) des Mikrokontrollers wird die „Start"-Sequenz und die Initialisierung ausgeführt. Darin werden wichtige Parameter deklariert. Daraufhin folgt die setup()-Funktion. Sie wird nur einmal durchgeführt und bestimmt z. B. welche Pins des Mikrokontrollers als Ausgänge oder Eingänge wirken. Sodann folgt eine while(true)-Schleife. In der Arduino-Umgebung wird diese als loop()-Funktion realisiert. Damit wird verhindert, dass der ATMEGA328P irgendwann zu einer „Stop"-Sequenz gelangen würde und man ihn neustarten müsste.

Nun durchläuft der Programmcode immer wieder einige wichtige Phasen. Zuerst wird der GPS-Chip hochgefahren, indem man einen 100ms langen Aus-An-Aus Impuls über den Pin mit dem Namen „GPS_ONOFF" ausgibt. [1, S. 23]

Einige Sekunden später erhält der Mikrokontroller die aktuellen Positionsdaten über eine UART Verbindung vom ORG1410. Diese Daten werden temporär in einem Puffer gespeichert und während der nächsten Phase, den Beschreiben der SD-Karte, gesichert. Im Anschluss folgen die nächsten Phasen: Herunterfahren des GPS-Chips und der Beginn der Tiefschlafphase des ATMEGA328P. In dieser Phase ist der Stromverbrauch am Geringsten. Erst 20 Minuten später wacht der Mikrokontroller durch eine „Interrupt"-Sequenz wieder auf und der Kreislauf beginnt erneut.

5 Platine des Prototypens

Das Layout wurde mithilfe der öffentlich zugänglichen Entwicklungssoftware „KiCad“ der Version 5.1.9 erstellt.

5.1 Abmessungen

Die ersten Prototypen der Platine sollen in einem 3d-gedruckten Gehäuse verbaut werden. Es sind möglichst kleine Abmessungen angestrebt, wobei die LiPo Batterie, der Mikrokontroller und der GPS-Empfangschip die groben Mindestmaße angeben. Diese betragen deswegen 60mm x 40mm x 30mm. Jedoch ist davon auszugehen, dass im Laufe der weiteren Entwicklung diese Maße weiter schrumpfen werden.

Es besitzt 2 Layer und normale Durchkontaktierungen (in Form von Nieten), weil die Anfertigung mithilfe einer CNC-Fräse gelingt.

5.2 Bauteilplatzierungen

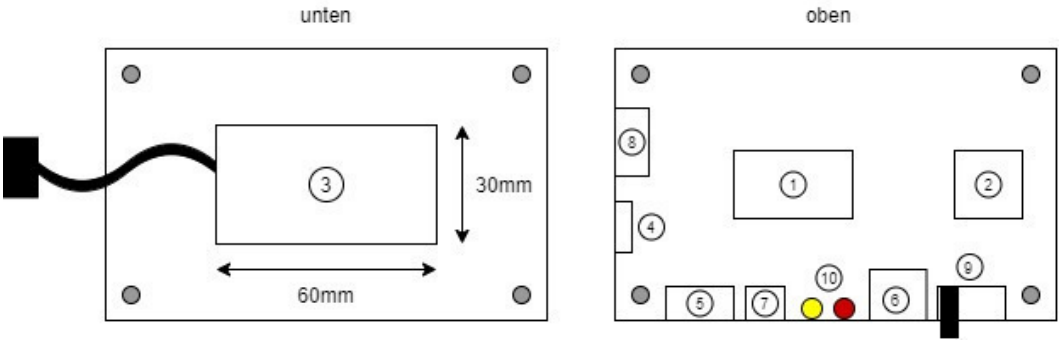


Abb. 7: Aufbau des Empfängers, Quelle: Eigene Darstellung

1: 8-bit Mikrokontroller
2: GPS Empfangschip
3: 3,7V Lithium Polymer Batterie
4: Thermistor Anschluss
5: Micro USB-Stecker
6: Micro SD-Karten Stecker
7: Druckschalter
8: Anschluss Batterie
9: Stromschalter
10: Leuchtdioden

Die Platzierung der Bauelemente auf der Platine ist der erste und einer der kritischsten Schritte beim Anfertigen eines Layouts. Deswegen wurde als erstes der Mikrokontroller als zentrales Element mittig platziert und so rotiert, dass alle Bus-Anbindungen möglichst ohne Durchkontaktierungen und mit sehr kurzer Länge geschaffen werden konnten. Der SPI Bus für die SD-Karte, als auch der I²C Bus für die Real Time Clock haben feste Pin-Vorgaben. Deswegen wurde als nächstes der GPS Empfangschip rechts davon platziert. Laut dem Hersteller sollte dieser möglichst isoliert auf der Platine angeordnet sein.

Alle Elemente, die für den Benutzer wichtig sind, sollten an der unteren Seite des Boards platziert werden, damit später im Gehäuse für diese nur eine einzige Aussparung nötig ist. Die Batterie befindet sich hinter der Hauptplatine in einer weiteren Aussparung.

5.3 Das Layout

Abbildung 8 zeigt als Endresultat das fertige Layout der beiden Layer: Rot ist die obere und Grün ist die untere Massefläche. Beide Flächen sind über mehrere Vias (engl. Durchkontaktierungen) miteinander verbunden.

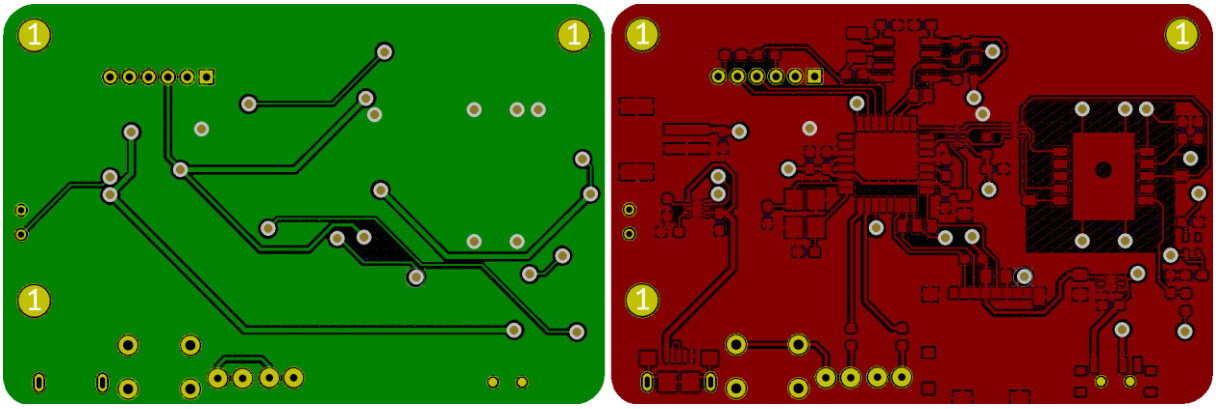


Abb. 8: Fertiges Layout mit einziger Massefläche, Quelle: Eigene Darstellung

Laut dem Hersteller des ORG1410 muss für einen reibungslosen Betrieb des Chips um ihn herum eine kupferfreie Fläche eingeplant werden. Diese betrifft auch die Unterseite der Antenne (rot/schwarz schraffierte Fläche, rechts in Abb. 7). Nur die kontaktierende Kupferfläche direkt unterhalb des Chips soll stehen bleiben und dient als eigene Masseinsel. Weil aber die verwendeten Masse-Vias zu breit und zu hoch sind für die individuelle Massefläche des Chips, mussten diese außerhalb platziert werden. Zudem sollte vermieden werden, dass man Signalleitungen auf der Unterseite des Chips verlegt.

Der Logikwandler befindet sich so nahe an dem Mikrokontroller, sodass der große 10µF Entkoppelkondensator des 3.3V Pegels auch für denselben Pegel des Logikwandlers verwendet werden kann.

6 Aufbauhinweise

6.1 Bestückung

Die Platine muss per Hand bestückt werden. Um Platz zu sparen wurde für jedes Bauteil darauf geachtet, dass eine SMD-Variante mit kleinem Formfaktor ausgewählt wurde. Jedoch gibt es auch Komponenten auf dem Board, die nur als Through-Hole-Variante verfügbar sind. Dafür sind diese aber groß genug, um von Studenten verlötet werden zu können. Das Problematischste Bauteil ist der GPS-Empfangschip. Jener darf im Lötprozess nicht zu lange erhitzt werden, da sonst die Gefahr besteht, dass er beschädigt oder zerstört wird.

Deshalb sollten die meisten Bauteile von einem Experten aufgelötet werden.

6.2 Inbetriebnahme

Bevor die Arduino-Firmware auf dem Mikrokontroller übertragen werden kann, muss ein passender „Bootloader“ in den Flash-Speicher des Mikrokontrollers gebrannt werden. Dies geschieht in der integrierten Arduino-Entwicklungsumgebung (kurz: IDE) über: Werkzeuge >> Bootloader brennen (siehe Abb. 9). Erst dann ist die serielle Schnittstelle verfügbar.

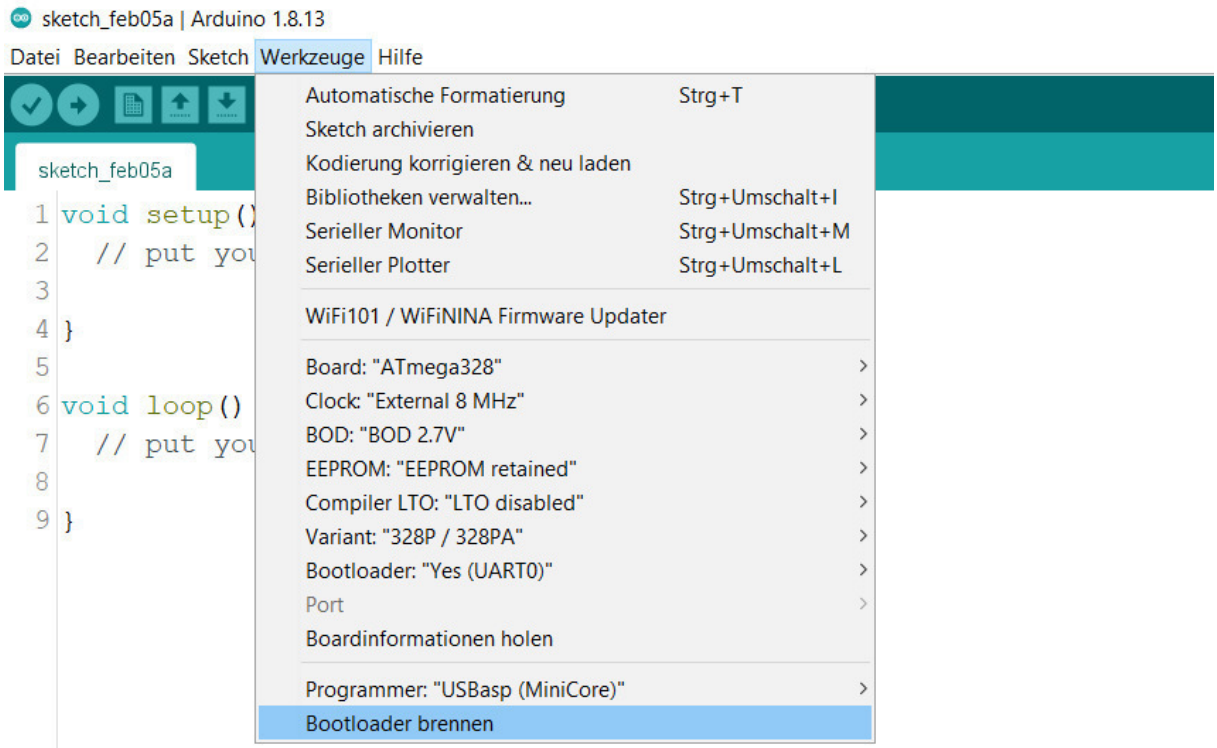
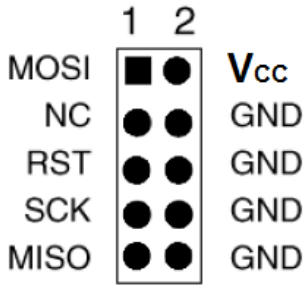


Abb. 9: Entwicklungsoberfläche und Boardeinstellungen, Quelle: Eigene Darstellung

Zuvor müssen jedoch die Boardeinstellungen auf dem richtigen Mikrokontroller-Chip eingestellt und ein Programmer (engl. Programmierer) ausgewählt werden. Hier wird ein

„USBasp“ als Programmer-Device im 3.3V Modus verwendet. Die Verbindungen vom Programmer zum Mikrokontroller sind in der unteren Tabelle (Abb. 10) verdeutlicht.



Programmer		Mikrokontroller
MOSI	➔	MOSI
MISO	➔	MISO
SCK	➔	SCK
Vcc	➔	3V3
GND	➔	GND
RST	➔	DTR

Abb. 10: Anschluss und Verbindungen des Programmers, Quelle: Eigene Darstellung

Ist dies abgeschlossen, so kann man nun mit dem Upload der Firmware beginnen. Der Upload erfolgt über die serielle Schnittstelle der Arduino-IDE. Da aber aus Platzgründen kein USB-zu-UART-Wandler auf die Platine verbaut wurde, muss ein externer Wandler für den Upload und das Kommunizieren mit dem Mikrokontroller (z.B. für Debugging-Nachrichten) benutzt werden. Hier wird ein FTDI232 als Wandler verwendet. Die Verbindungen vom Wandler zum Mikrokontroller sind in der unteren Tabelle (Abb. 11) verdeutlicht.

USB-UART-Wandler		Mikrokontroller
TX	➔	RX
RX	➔	TX
Vcc	➔	3V3
CTS	➔	CTS oder GND
GND	➔	GND
DTR	➔	DTR

Abb. 11: Anschluss und Verbindungen des FTDI232, Quelle: Eigene Darstellung

7 Ausblick und Verbesserungsvorschläge

Dieses Projekt bietet so viel Entwicklungspotential, sodass es noch weiterbearbeitet werden sollte. Vor allem der Dateiaustausch über die SD-Karte könnte noch weiter verbessert werden, indem man ein Bluetooth-Modul verwendet und die gespeicherten Positionsinformationen kabellos zu einem Computer oder einem Handy mit einer passenden „Android-App“ überträgt. Alternativ könnte auf einen Mikrokontroller gewechselt werden, der nativ eine USB-Verbindung erlaubt. Ein Beispiel hierfür wäre der ATMEGA 32u4 von Atmel oder ein STM32 aus der Serie L1.

Darüber hinaus könnte beim 3,3 Volt Pegel auf einen Schaltregler mit höherer Effizienz gewechselt werden.

Literaturverzeichnis

[1] J.-M. Zogg, GPS und GNSS: Grundlagen der Ortung und Navigation mit Satelliten, ublox AG, 2001.

[2] Atmel, „Datenblatt des ATMEGA328P," Januar 2015. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. [Zugriff am 1. Februar 2021].

[3] Texas Instruments, „Datenblatt des LSF0204," November 2019. [Online]. Available: https://www.ti.com/lit/ds/symlink/lsf0204.pdf. [Zugriff am 1. Februar 2021].

[4] Texas Instruments, „Datenblatt des TPS7178DCKR," Januar 2016. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps717.pdf. [Zugriff am 1. Februar 2021].

[5] Texas Instruments, „Datenblatt des TPS74533PQWDRQ1," Januar 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps745-q1.pdf. [Zugriff am 1. Februar 2021].

[6] R. Cerda, „Application Note der Firma Crystek, Pierce-gate oscillator crystal load calculation," Juli 2004. [Online]. Available: https://www.crystek.com/documents/appnotes/PierceGateLoadCap.pdf. [Zugriff am 1. Februar 2021].

[7] OriginGPS, „Application Note zum ORG1410," 25. August 2019. [Online]. Available: https://origingps.com/wp-content/uploads/2019/09/HORNET-Modules-Layout-recommendations-and-Integration-Application-Note-Rev-2.0.pdf. [Zugriff am 1. Februar 2021].

[8] OriginGPS, „Datenblatt des Hornet ORG1410," 1. Dezember 2020. [Online]. Available: https://origingps.com/wp-content/uploads/2021/01/Micro-Hornet-ORG1410-Datasheet-4.4.pdf. [Zugriff am 1. Februar 2021].

[9] Vishay Semiconductors, „Datenblatt des VEMI45AA," 1. Januar 2021. [Online]. Available: https://www.vishay.com/docs/81385/vemi45aa.pdf. [Zugriff am 1. Februar 2021].

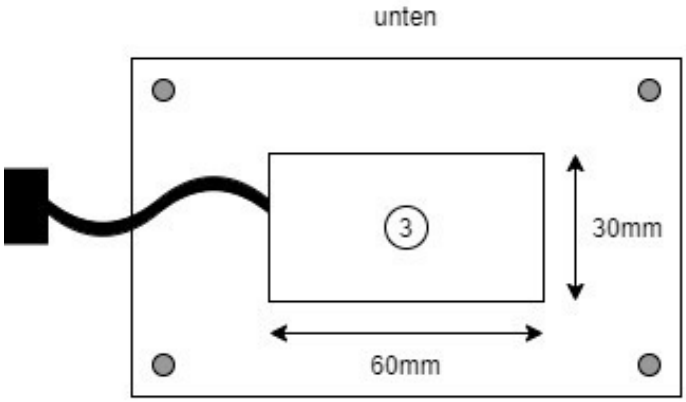
Anhang A: Datenblatt des Empfängers

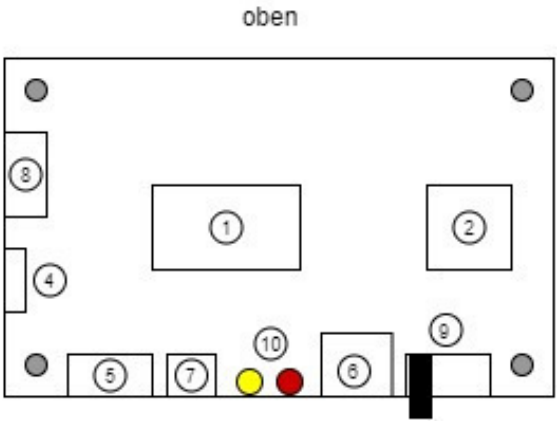
„Track your Cat“ - GPS-Empfänger

1 Merkmale

- Keine monatlichen Gebühren
- Kleine Dimensionierung
- Lange Batterielaufzeit
- Plug & Play
    - Einfach zu verwendendes Gerät
    - Datenformat passend für Google Earth
- Niedriges Gewicht
    - Entlastend für Haustiere
- Gehäuse passend für gängige Halsbänder
- Micro-SD Karte
    - günstiger Speicher

2 Aufbau





1: 8-bit Mikrokontroller

2: GPS Empfangschip

3: 3,7V Lithium Polymer Batterie

4: Thermistor Anschluss

5: Micro USB-Stecker

6: Micro SD-Karten Stecker

7: Druckschalter

8: Anschluss Batterie

9: Stromschalter

10: Leuchtdioden

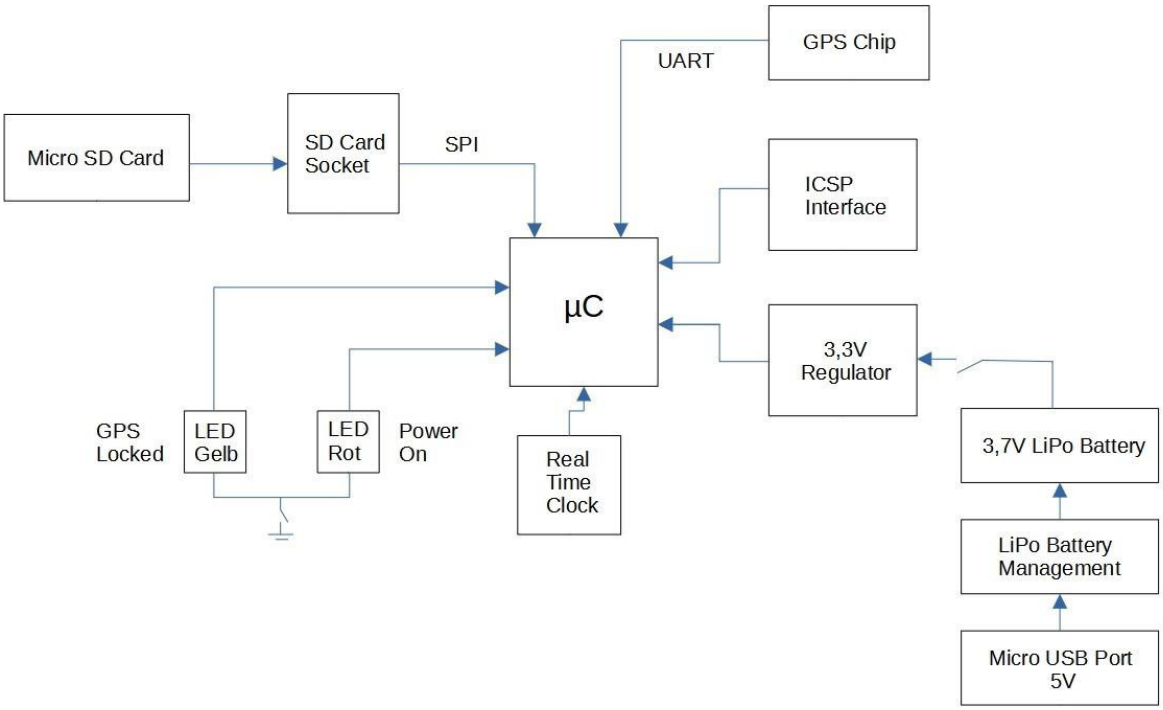
3 Beschreibung

Das „Track your Cat“ ist ein GPS-Empfangsgerät basierend auf einen 8-bit Mikrocontroller von Atmel mit wieder-aufladbarem Akku. Als Receiver Chip wird der ORG1410 von OriginGPS verwendet.
Es besteht aus einer kompakten Hauptplatine und ist in einem 3D-gedrucktes Gehäuse eingebaut. Die Empfangsantenne ist Innen im Gehäuse integriert und somit vor äußeren Umwelteinflüssen geschützt.

4 Anwendungen

- Haustiere verfolgen
- Eigenes Auto verfolgen
- Joggingstrecke verfolgen

5 Block-Diagramm

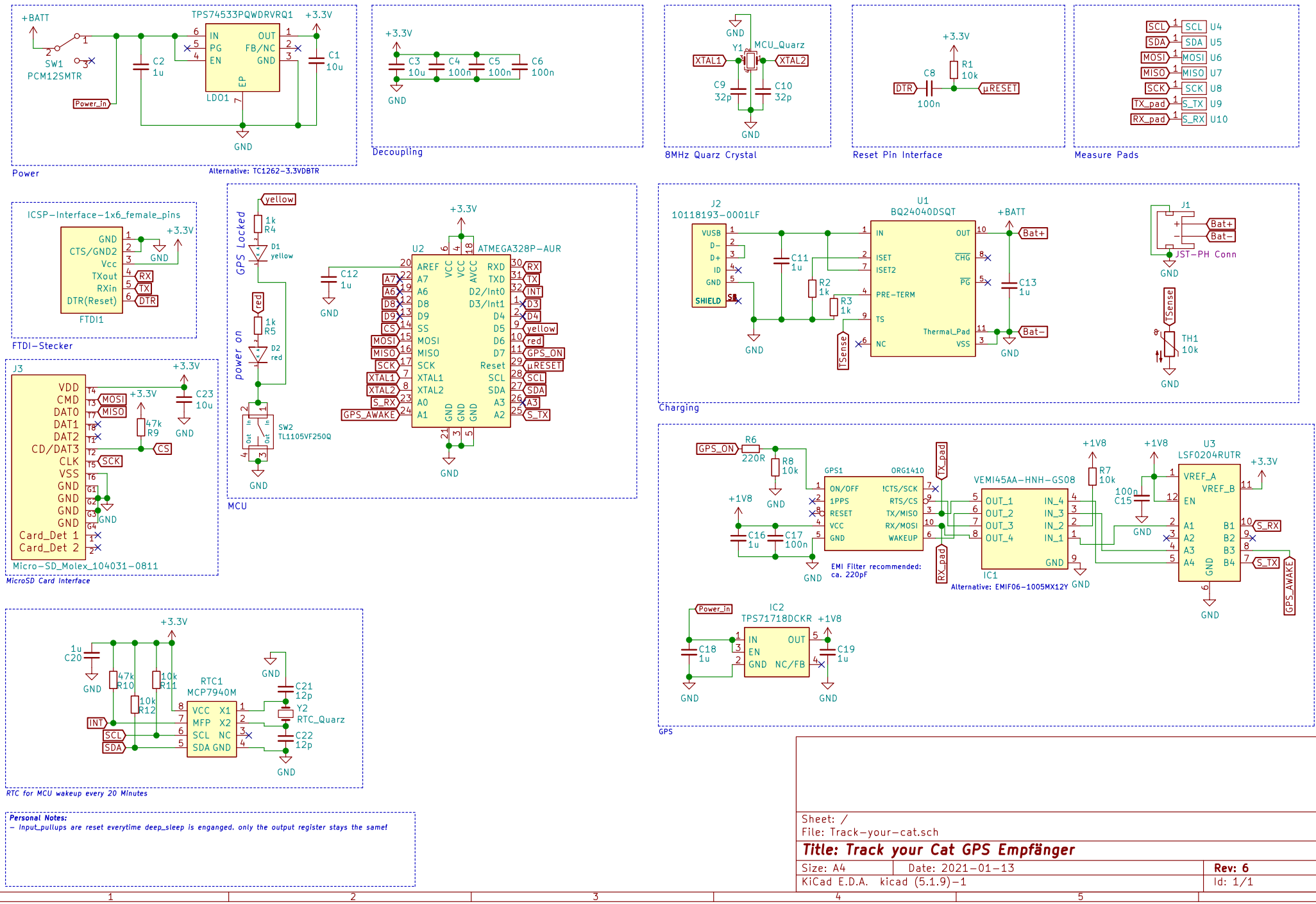


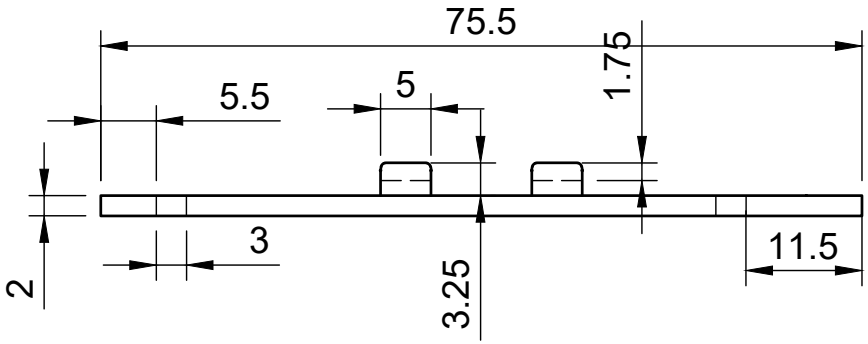
6 Bestückungsliste

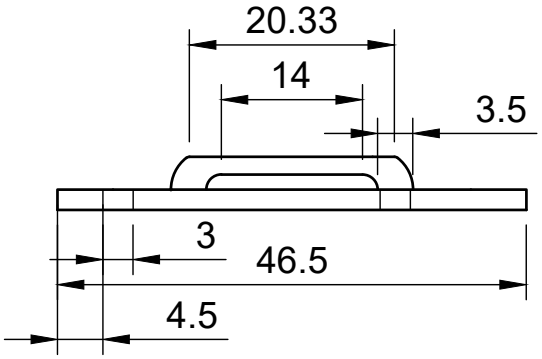
Referenz	Name	Wert	Mouser Teile Nummer	Anzahl
C1, C3, C23	Kondesator	10u		2
> C4-C8, C14, C15, C17	Kondesator	100n		8
> C21, C22	Kondesator	12p		2
> C9, C10	Kondesator	32p		2
> C2, C11-C13, C16, C18-C20	Kondesator	1u		8
D1	LED	yellow	---	1
D2	LED	red	---	1
FTDI1	Programmer	ICSP-Interface-1x6_female_pins	---	1
GPS1	GPS-IC	ORG1410	---	1
IC1	EMI Filter	VEMI45AA-HNH-GS08	78-VEMI45AA-HNH	1
IC2	Regulator	TPS71718DCKR	---	1
J1	JST-PH Conn	JST-PH 2-pin SMT right angle	adafruit 1769	1
J2	Micro USB typ B	USB_B_Micro	Amphenol 10118193-0001LF	1
J3	MicroSD Slot	Micro-SD_Molex_104031-0811	538-104031-0811	1
LDO1	Regulator	TPS74533PQWDRVRQ1	595-PS74533PQWDRVRQ1	1
> R1, R7, R8, R11, R12	Widerstand	10k		5
> R2-R5	Widerstand	1k		4
R6	Widerstand	220R		1
> R9, R10	Widerstand	47k		2
RTC1	RTC	MCP7940M	579-MCP7940MT-I/SN	1
SW1	Schalter	PCM12SMTR	611-PCM12SMTR	1
SW2	Schalter	TL1105VF250Q	612-TL1105VF250Q	1
TH1	Thermistor	10k	NXFT15XH103FA2B050	1
U1	Lade IC	BQ24040DSQT	595-BQ24040DSQT	1
U2	MCU	ATMEGA328P-AUR	556-ATMEGA328P-AU	1
U3	Logic shifter	LSF0204RUTR	---	1
Y1	Crystal	MCU_Quarz	ABM8AIG-8.000MHZ-1Z-T	1
Y2	Crystal	RTC_Quarz	ECS-.327-12.5-34QN-TR	1

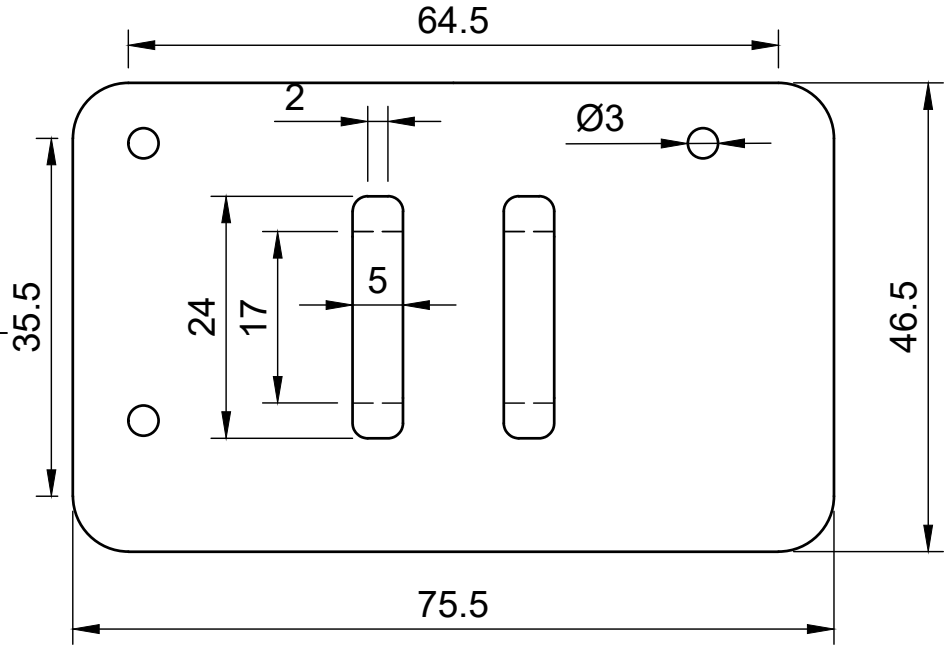
Anhang B: CAD-Zeichnungen

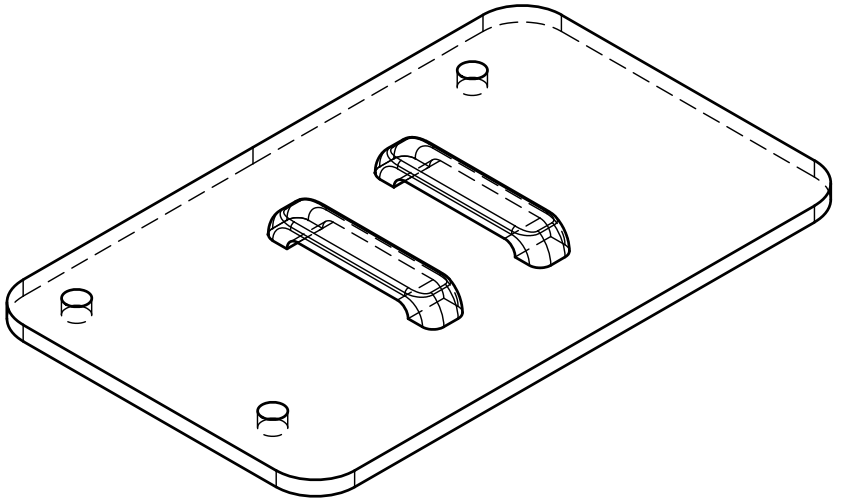
- Schaltplan
- Gehäuse-Zeichnungen


Power
Alternative: TC1262-3.3VDBTR
Decoupling
8MHz Quarz Crystal
Reset Pin Interface
Measure Pads
FTDI-Stecker
MCU
Charging
MicroSD Card Interface
GPS
RTC for MCU wakeup every 20 Minutes
Personal Notes:
- Input_pullups are reset everytime deep_sleep is enganged. only the output register stays the same!
Sheet: /
File: Track-your-cat.sch
Title: Track your Cat GPS Empfänger
Size: A4
Date: 2021-01-13
Rev: 6
KiCad E.D.A. kicad (5.1.9)-1
Id: 1/1

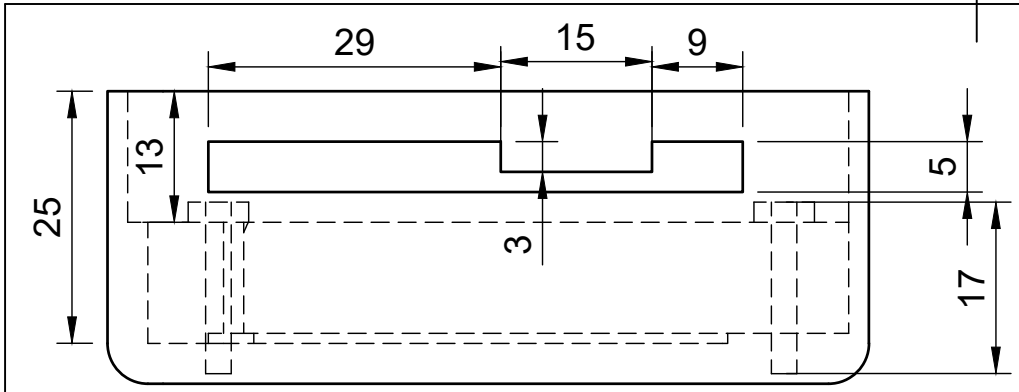


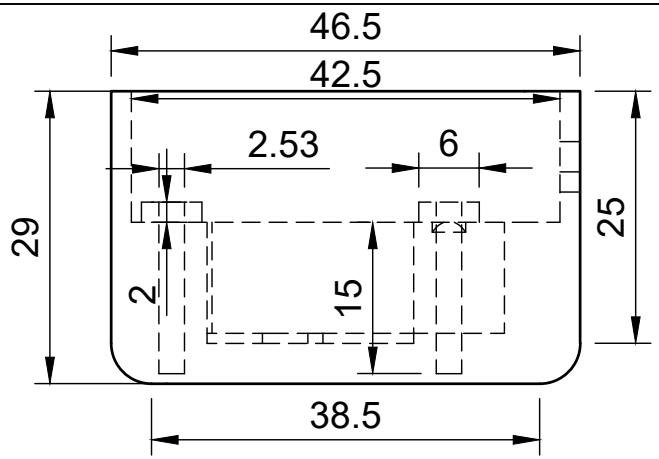


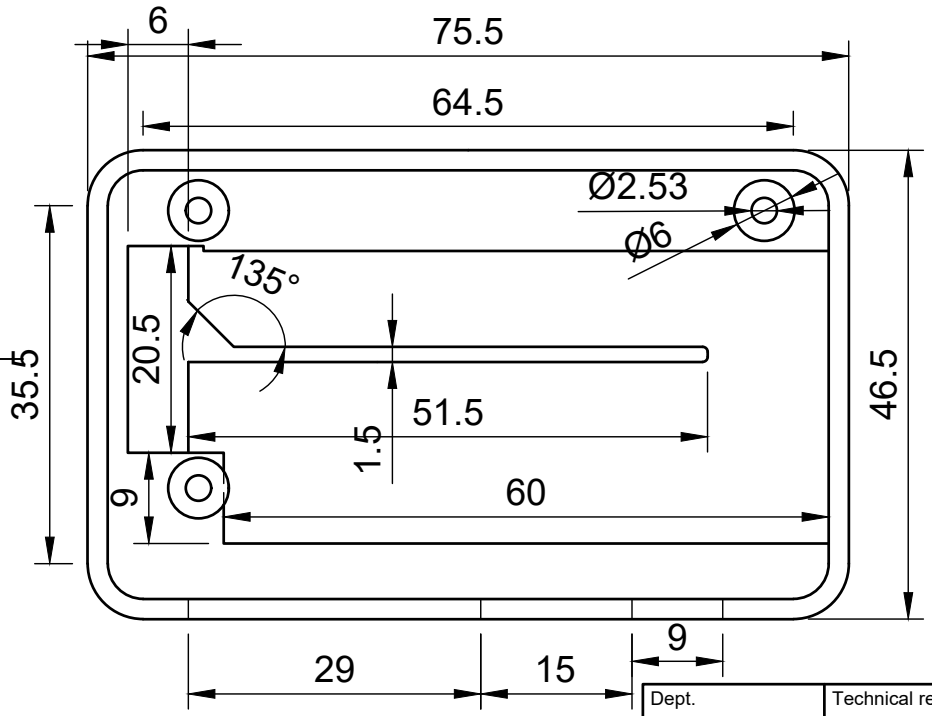


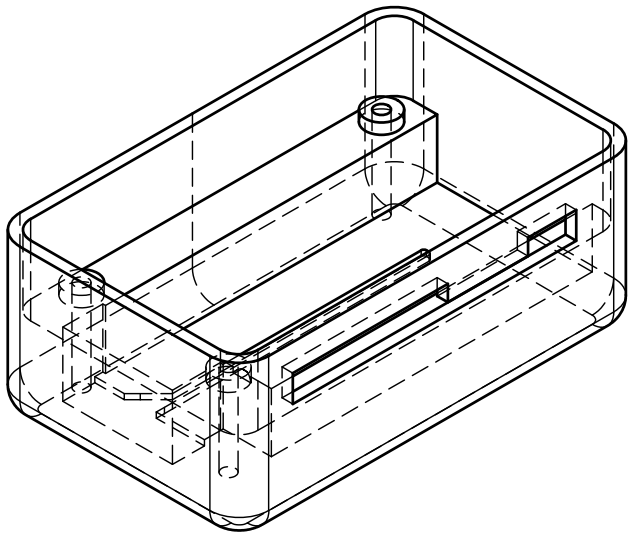


Dept.	Technical reference	Created by Patrick Gröppner 04.02.2021	Approved by
		Document type	Document status
		Title Gehäuse Deckel	DWG No.
			Rev. / Date of issue / Sheet 1/1









Dept.	Technical reference	Created by Patrick Gröppner 04.02.2021	Approved by
		Document type	Document status
		Title Gehäuse Basis	DWG No.
			Rev. / Date of issue / Sheet 1/1

Anhang C: Programmcode des Mikrokontrollers

```
/* This program will write GPS NMEA data to a micro SD card every X minutes */
#include <Wire.h>
#include <LowPower.h>
#include <MCP7940.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <SD.h>
//#include <TinyGPS++.h> /* used to decode the nmea messages into timestamps and geografical coordinates */

// The Arduino pins used
static const int GPS_ONOFFPin = 7;
#define GPS_RXPin A0
#define GPS_TXPin A2
#define GPS_SYSONPin A1
static const int ISRwakeUpPin = 2; // pin 2 is wake up pin = INT0
static const int chipSelect = 10;
static const int LEDyellow = 5;
static const int LEDred = 6;

boolean mfpPinTriggered = true; //for single loop after startup

// How often should the positions be written to the sd card after every x minutes?
const uint8_t writeIterations = 200;

static const uint8_t GPSBaud = 4800;
static const uint32_t serial_baud = 115200;     //Set the baud rate for Serial I/O comm
const uint8_t  SPRINTF_BUFFER_SIZE = 32;  //Buffer size for sprintf()
const uint8_t  ALARM_INTERVAL = 20;       //Interval of X min

/* Enumeration of MCP7940 alarm types */
enum alarmTypes {
  matchSeconds,
  matchMinutes,
  matchHours,
  matchDayOfWeek,
  matchDayOfMonth,
  Unused1,
  Unused2,
  matchAll,
  Unknown
  };

MCP7940_Class MCP7940;
char inputBuffer[SPRINTF_BUFFER_SIZE];  //Buffer for sprintf() or sscanf()

// The GPS connection is attached with a software serial port
SoftwareSerial Gps_serial(GPS_RXPin, GPS_TXPin);

int inByte = 0; // incoming serial byte
byte pbyGpsBuffer[100];
int byBufferIndex = 0;

void wakeUp() {
    //handler for the pin interrupt.
    mfpPinTriggered = true;
    // Disable external pin interrupt on wake up pin: prevent multiple wakeups
    detachInterrupt(0);
}
```

```
/***********************
********SETUP*********
***********************/
void setup()
{
  // Open the debug serial port
  Serial.begin(serial_baud);

  // Open the GPS serial port
  Gps_serial.begin(GPSBaud);

  pinMode(GPS_SYSONPin, INPUT);
  pinMode(GPS_ONOFFPin, OUTPUT);
  digitalWrite(GPS_ONOFFPin, LOW );
  delay(5);
  pinMode(ISRwakeUpPin, INPUT);

  //led pinmodes
  pinMode(LEDyellow, OUTPUT);
  pinMode(LEDred, OUTPUT);
  digitalWrite(LEDyellow, LOW);
  digitalWrite(LEDred, LOW);

  //To save some extra power it is recommended to set every unused pin to OUTPUT mode
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(A3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(3, OUTPUT);

  pinMode(chipSelect, OUTPUT); // chip select pin set to output

  /* //Waking up the the GPS chip
  if (digitalRead(GPS_SYSONPin) == LOW)
  {
    digitalWrite(GPS_ONOFFPin, HIGH);
    delay(100);
    digitalWrite(GPS_ONOFFPin, LOW);
    delay(1000);
  }
  */
  while (!MCP7940.begin())  //Loop until the RTC communications are established
  {
    Serial.println(F("Unable to find MCP7940. Checking again in 3s."));
    delay(3000);
  }
  Serial.println(F("MCP7940 initialized."));
  while (!MCP7940.deviceStatus())  // Turn oscillator on if necessary
  {
    Serial.println(F("Oscillator is off, turning it on."));
    bool deviceStatus = MCP7940.deviceStart();  // Start oscillator and return state
    if (!deviceStatus) {
      Serial.println(F("Oscillator did not start, trying again."));
      delay(1000);
   }
  }

  Serial.println("Setting MCP7940M to date/time of library compile");
  MCP7940.adjust();  // Use the compilers date/time to set clock
  Serial.print("Date/Time set to ");
  DateTime now = MCP7940.now();  // get the current time
```

```
  // Use sprintf() to print date/time with leading zeroes
  sprintf(inputBuffer,  "%04d-%02d-%02d  %02d:%02d:%02d",  now.year(),  now.month(),
now.day(), now.hour(), now.minute(), now.second());
  Serial.println(inputBuffer);

  MCP7940.setAlarm(0, matchAll, now, true);

  // see if the card is present and can be initialized:
  Serial.print("Initializing SD card...");
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    return;
  }
  Serial.println("card initialized.");

  digitalWrite(LEDred, HIGH); // indicate power is on
}
/**********************
********LOOP***********
**********************/
void loop()
{
  static uint8_t secs;
  DateTime now = MCP7940.now();  // get the current time

  if (secs != now.second())      // Output if seconds have changed
  {
 //   sprintf(inputBuffer,  "%04d-%02d-%02d  %02d:%02d:%02d",  now.year(),  now.month(),
now.day(), now.hour(), now.minute(), now.second());
 // Serial.print(inputBuffer);
    secs = now.second();     // Set the counter for later comparision

    if (MCP7940.isAlarm(0))  // When alarm 0 is triggered
    {
    //Serial.print(" *Alarm1* resetting to go off next at ");
      now = now + TimeSpan(0, 0, ALARM_INTERVAL, 0);  // Add interval to current time
    //sprintf(inputBuffer,  "%04d-%02d-%02d  %02d:%02d:%02d",  now.year(),  now.month(),
now.day(), now.hour(), now.minute(), now.second());
      Serial.print(inputBuffer);
      MCP7940.setAlarm(0, matchAll, now, true);  // Set alarm to go off in 20 min again
    }
    Serial.println();
  }
  // Allow wake up pin to trigger interrupt on low.
  attachInterrupt(0, wakeUp, LOW);

  if (mfpPinTriggered == true) {

  mfpPinTriggered = false;
  };
  // Enter power down state with ADC and BOD module disabled.
  // Wake up when wake up pin is low.
  LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);

  if (!SD.begin(chipSelect)) {
    return;
  }
  //Wakeup the GPS chip and wait for a fix
  if (digitalRead(GPS_SYSONPin) == LOW)
  {
  digitalWrite( GPS_ONOFFPin, HIGH );
```

```
  delay(100);
  digitalWrite( GPS_ONOFFPin, LOW );
  while (digitalRead(GPS_SYSONPin) == LOW){
  delay(1000);
  };
for(int k=1; k <= writeIterations; k++) {
  byte byDataByte;
  if (Gps_serial.available())
  {
     byDataByte = Gps_serial.read();

     Serial.write(byDataByte);
     pbyGpsBuffer[byBufferIndex++] = byDataByte;

     if(byBufferIndex >= 100)
     {
       byBufferIndex = 0;
       File dataFile = SD.open("gps.txt", FILE_WRITE);
       // if the file is available, then write to it:
       if (dataFile) {
        dataFile.write(pbyGpsBuffer, 100);
        dataFile.close();
       }
       // if the file isn't open, pop up an error:
       else {
         Serial.println("error opening gps.txt");
       }
     }
    }
  }
  //Turn off the GPS chip and then wait some seconds
  if (digitalRead(GPS_SYSONPin) == HIGH)
  {
  digitalWrite(GPS_ONOFFPin, HIGH);
  delay(100);
  digitalWrite(GPS_ONOFFPin, LOW);
  while (digitalRead(GPS_SYSONPin) == HIGH){
    delay(500);
  }
 }
}
}
```