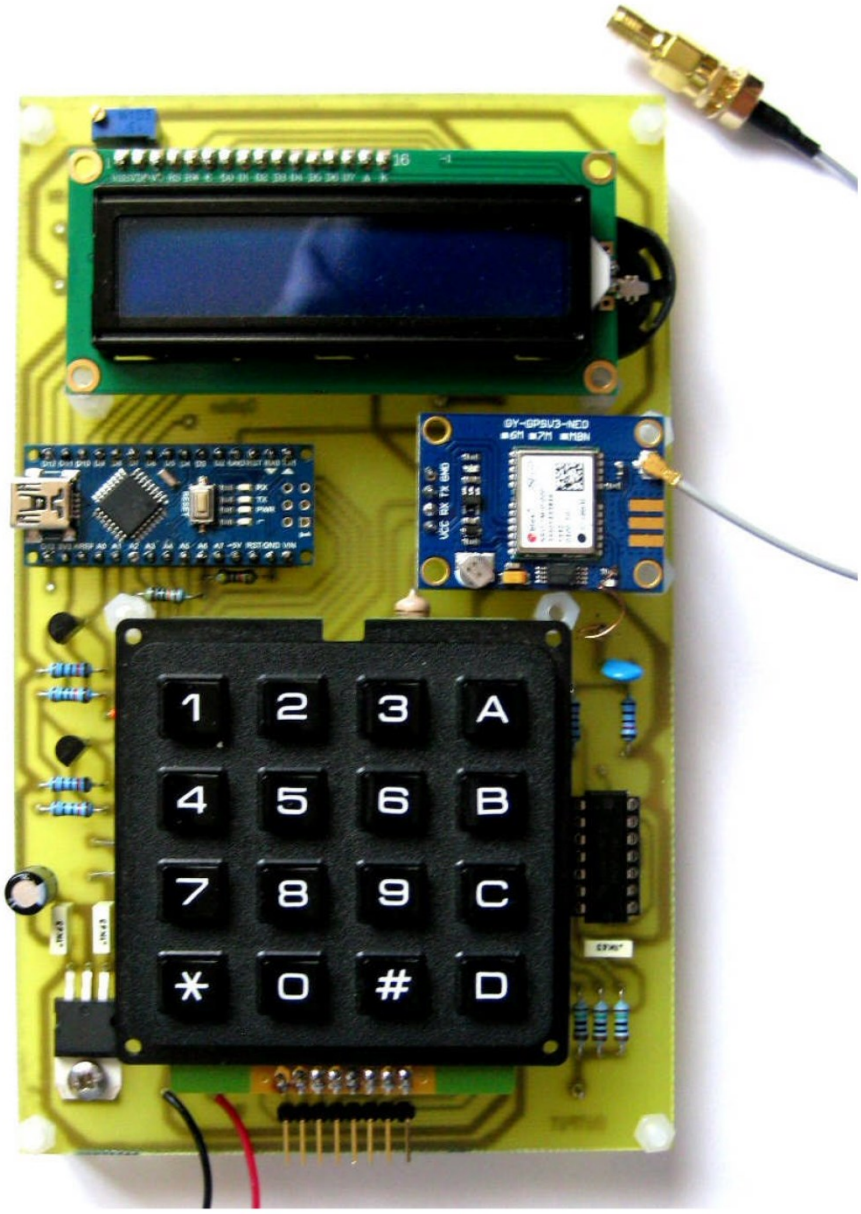
SCULLCOM
Hobby Electronics



GPS Locked

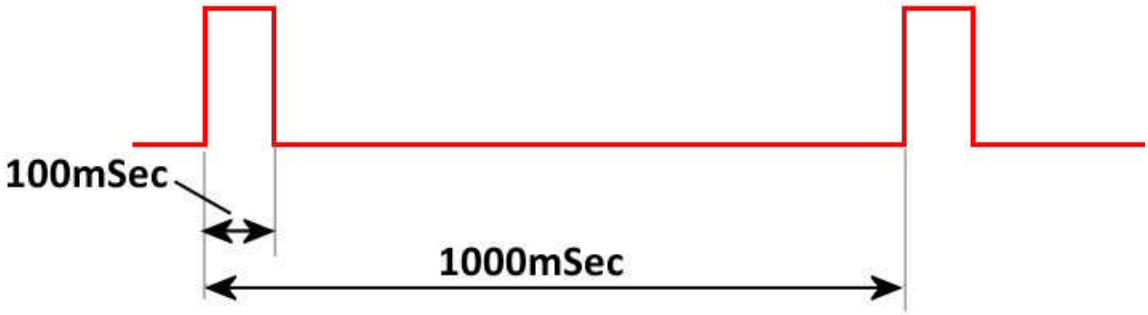
Frequency Reference Standard

(Keypad Version)

Construction Manual (updated)

Projekt für GPS-genaue Standardfrequenzen

In diesem Projekt wird ein kleines GPS-Modul verwendet, das über den GPS-Empfänger u-blox Neo-7 verfügt. Diese Module sind auf Websites wie eBay für etwa 12 Euro erhältlich und werden in der Regel für die Verwendung mit Fluglotsen beworben auf Modellflugzeugen.
Der U-Blox-Empfänger befindet sich normalerweise in der Mitte einer kleinen Leiterplatte. Es befinden sich zusätzliche Komponenten auf der Platine.
Dazu gehören ein 3,3-Volt-Low-Drop-Spannungsregler, eine Zeitimpuls- und die Einschalt-LED sowie das EEPROM. Es gibt auch einige kurzfristige Backup-Versorgung für den DRAM des Neo-7-Empfängers unter Verwendung eines 0,08F-Superkondensators (Seiko XH414HG). Diese Kondensatoren liefern normalerweise nur für etwa einen Tag eine Notstromversorgung, wenn Sie dies überhaupt schaffen.
Diese GPS-Module sind normalerweise standardmäßig so konfiguriert, dass sie einen 1-Hz-Zeitimpulsausgang liefern. Dieser wird normalerweise verwendet, um mit einer grünen, blinkenden LED anzuzeigen, wann das Gerät einen Fix für Satelliten hat und der lokale Oszillator mit dem GPS-Signal verkoppelt ist. Standardmäßig liefert dieser Zeitsteuerungsimpuls alle 1000 ms einen Impuls von 100 ms, wie unten gezeigt.



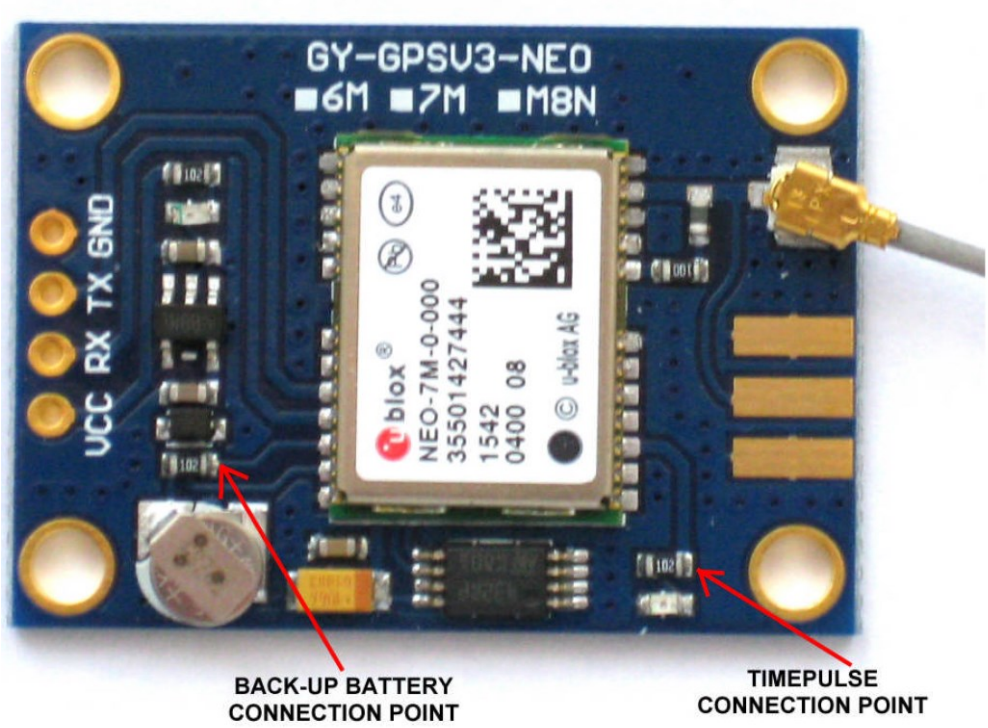
Dieser Zeitsteuerungsimpuls kann konfiguriert werden, um unterschiedliche Frequenzen und unterschiedliche Arbeitszyklen bereitzustellen, und wir werden diese Option in diesem Projekt verwenden, um die Frequenz zu ändern und den Arbeitszyklus auf 50% festzulegen.
GPS-Satelliten werden hauptsächlich als Navigationssystem verwendet. Dieses globale Positionierungssystem kann auch verwendet werden eine genaue Zeit, Zeitintervalle und Häufigkeit zu verbreiten. Die GPS-Trägersignale stammen von einem Bordgerät Atomuhr (Oszillator), die von Bodenstationen in den USA überwacht und gesteuert werden.
Sie stellen sicher, dass es mit der koordinierten Weltzeit (UTC) übereinstimmt. UTC ist der primäre Zeitstandard, nach dem die Welt Uhren und Zeit reguliert.
Atomuhren basieren auf den natürlichen atomaren Schwingungen von Gasen in Resonanzhohlräumen. Isoliert von Magnetfeldern schwingen Rubidium- und Cäsiumgase bei bestimmten Frequenzen unter kontrollierten Bedingungen. Diese Frequenzen sind so genau, dass seit 1967 die Länge der Sekunde definiert ist als die Frequenz eines bestimmten Resonanzmode des Cäsiumatoms, die 9.192.631.770 Schwingungen in einer Sekunde erzeugt.

Der U-Blox-Empfänger verwendet einen eingebauten 48-MHz-Oszillator.
Es gibt verschiedene Versionen des U-Blox-Empfängers, von denen einige so konfiguriert werden können, dass sie eine Frequenzimpulsausgabe im Bereich von 0,25 Hz bis 10 MHz erzeugen.

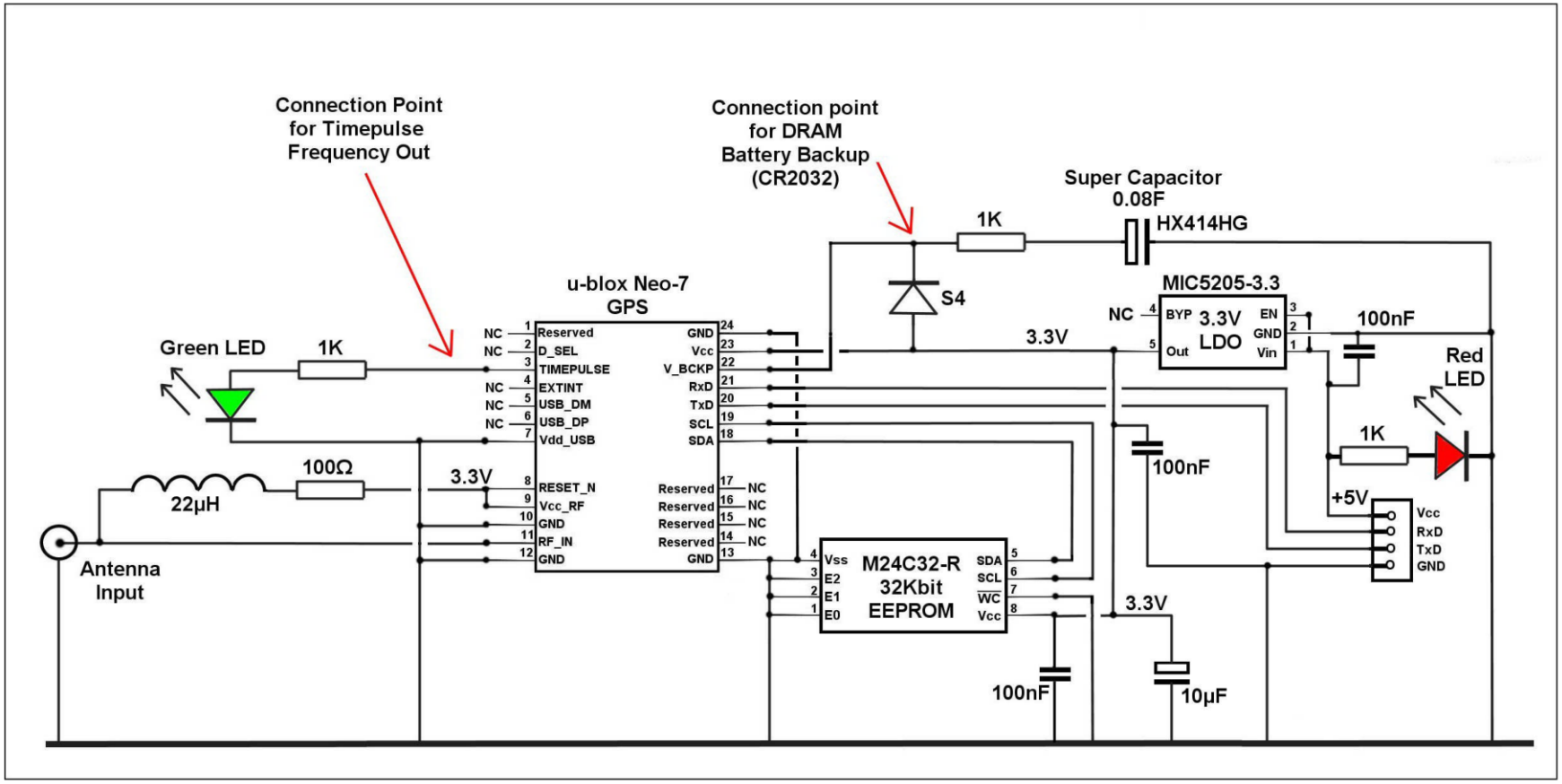
Diejenigen, die für dieses Projekt geeignet sind, sind in der nachstehenden Tabelle aufgeführt.

u-blox Empfängertyp	Art des Speichers	Typ des verwendeten Oszillators
Neo-6T	ROM	TCXO
Neo-7M	ROM	Crystal
Neo-7N	FLASH	TCXO
Neo-7P	FLASH	Kristall
Neo-M8M	ROM	Crystal
Neo-M8N	FLASH	TCXO
Neo-M8Q	ROM	Crystal

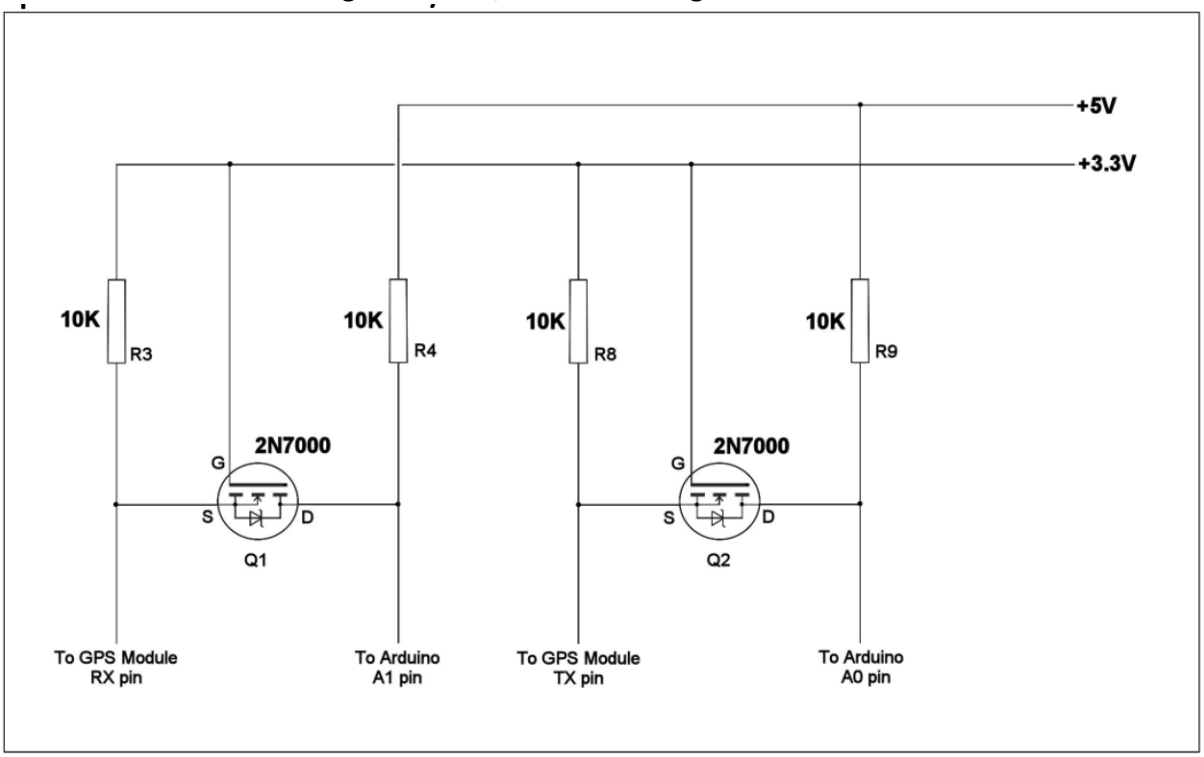
In diesem Projekt werden wir den Neo-7M verwenden. Da hierfür ein ROM verwendet wird, besteht die Möglichkeit, eine 3-Volt-Pufferbatterie zu verwenden.
Die vorgestellte Leiterplatte beschleunigt das Satelliteneinrasten nach dem Einschalten (dies ist optional). Es kann einige Minuten dauern, bis sich das Gerät einlockt wenn die Pufferbatterie nicht verwendet wird. Wenn die Neo-7N-Version mit FLASH-Speicher verwendet wird, ist keine Sicherungsversorgung erforderlich.



Erforderliche Verbindungspunkte:
Löten Sie ein Verbindungskabel an. Verwenden Sie Kabel mit geringer Stärke, um nicht zu viel zu Belastung der Modulplatinenpunkte zu erzeugen.


Connection Point
for Timepulse
Frequency Out
Connection point
for DRAM
Battery Backup
(CR2032)
Super Capacitor
0.08F
HX414HG
1K
S4
u-blox Neo-7
GPS
MIC5205-3.3
3.3V
LDO
BYP
Out
EN
GND
Vin
NC
100nF
Green LED
1K
3.3V
Red
LED
1K
100Ω
22µH
3.3V
100nF
+5V
Vcc
RxD
TxD
GND
Antenna
Input
Reserved
D_SEL
TIMEPULSE
EXTINT
USB_DM
USB_DP
Vdd_USB
RESET_N
Vcc_RF
GND
RF_IN
GND
Vcc
V_BCKP
RxD
TxD
SCL
SDA
Reserved
Reserved
Reserved
Reserved
NC
NC
NC
NC
M24C32-R
32Kbit
EEPROM
Vss
E2
E1
E0
SDA
SCL
WC
Vcc
3.3V
100nF
10µF

Betrieb der verwendeten Pegelwandler- / Schaltschaltung



Das u-blox Neo-7 GPS-Modul arbeitet mit 3,3 Volt und daher ist beim Anschließen der GPS serielle TX- und RX-Pins direkt an den Arduino, der mit 5 Volt betrieben wird, Vorsicht geboten. Obwohl in der Praxis, wenn nur der serielle GPS TX-Pin verwendet wird, scheint dies einwandfrei zu funktionieren, da der Arduino offenbar in der Lage ist, niedrigere Logik der Pegelausgänge vom GPS-Modul zu verarbeiten.
Wenn jedoch auch der GPS-RX-Pin mit dem Arduino verbunden ist, kann dies Schäden am GPS-U-Blox Neo-7 verursachen, da der Arduino ihn mit 5-Volt-Logikpegeln antreibt.
Daher ist es technisch korrekt, eine Logikpegelwandlerschaltung zu verwenden, um die 5-Volt-Logikpegel auf 3-Volt-Logikpegel zu verschieben.
Die obige Schaltung, die zwei N-Kanal-MOSFETs verwendet, bietet eine einfache Methode zum Konvertieren / Verschieben, diese logischen Ebenen und wir verwenden diese Technik in unserem Projekt.
Die Idee des Logikpegelwandlers gibt es schon seit einiger Zeit. In den späten 90er Jahren hatten Philips R & D - Ingenieure in den Niederlanden einige Probleme auf Logikebene mit ihrer I²C-Bus-Technologie und dort kamen die Ingenieure auf die Idee, einen 'N'-Kanal-MOSFET zu verwenden, um die logisch hohen Pegel zu verschieben. Die zeitliche Entwicklung der IC-Technologie verbesserte sich, was zu Abständen von weniger als 0,5 µm auf dem Siliziumchip führte, wodurch der logisch hohe Pegel und die Versorgungsspannungen auf etwa 3,3 Volt begrenzt wurden.
Dieser bidirektionale Pegelwandler / Shifter dient zum Anschließen der seriellen TX- und RX-Pins des GPS-Moduls an das Arduino mit jeweils unterschiedlicher Versorgungsspannung und unterschiedlichen Logikpegeln. Die obige Schaltung hat zwei Logik Level Shifter, beide identisch. Wir werden also die linke Seite des Diagramms betrachten. Die Source, des MOSFET, der an das GPS-Modul angeschlossen ist, verfügt über Pull-up-Widerstände, die an eine 3,3-Volt-Versorgung angeschlossen sind. Der Drain des MOSFET, der mit dem Arduino verbunden ist, hat Pull-up-Widerstände, die an 5 Volt angeschlossen sind.

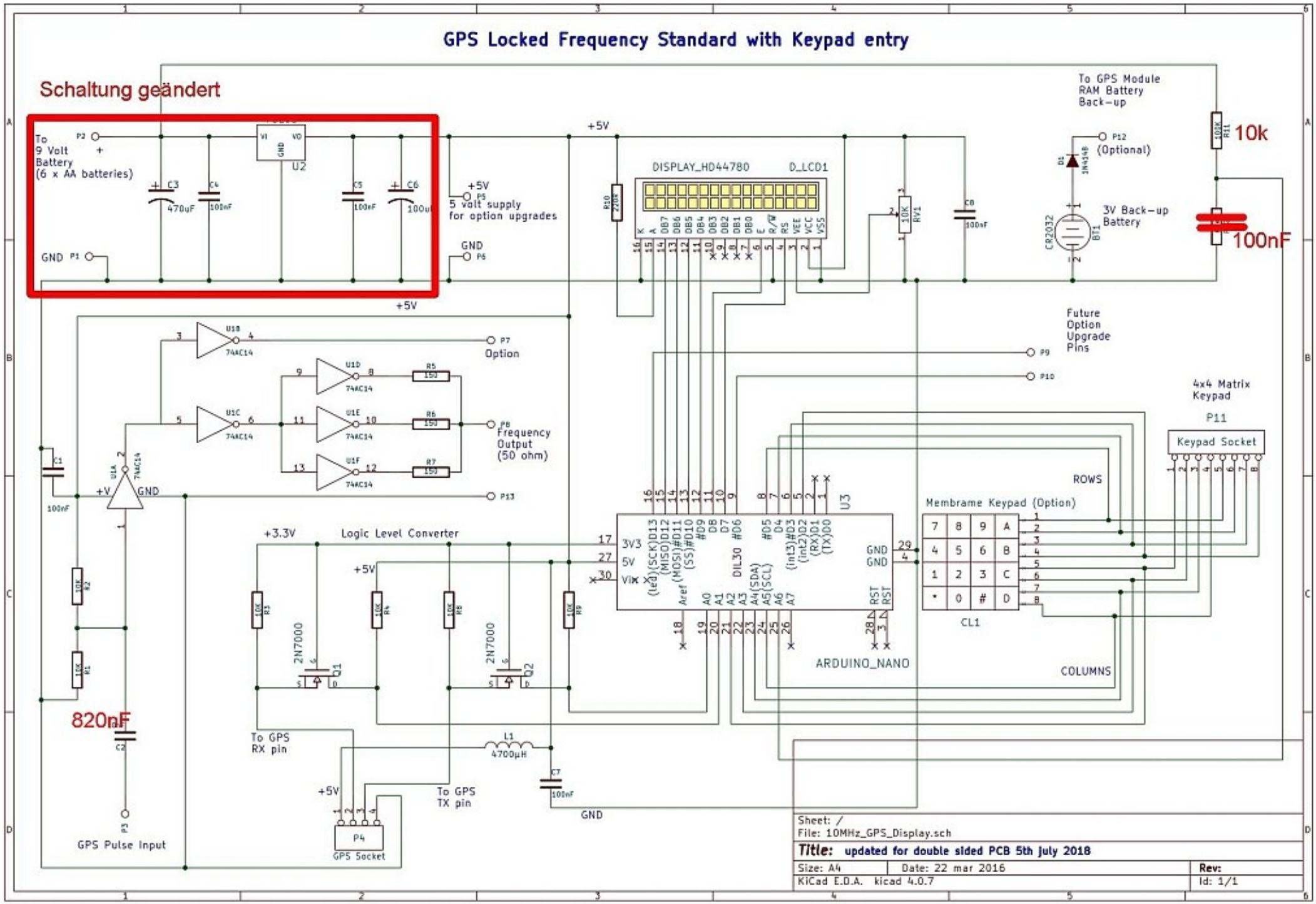
Die folgenden drei Zustände sollten während des Betriebs dieses Pegelwandlers / Schalters berücksichtigt werden:

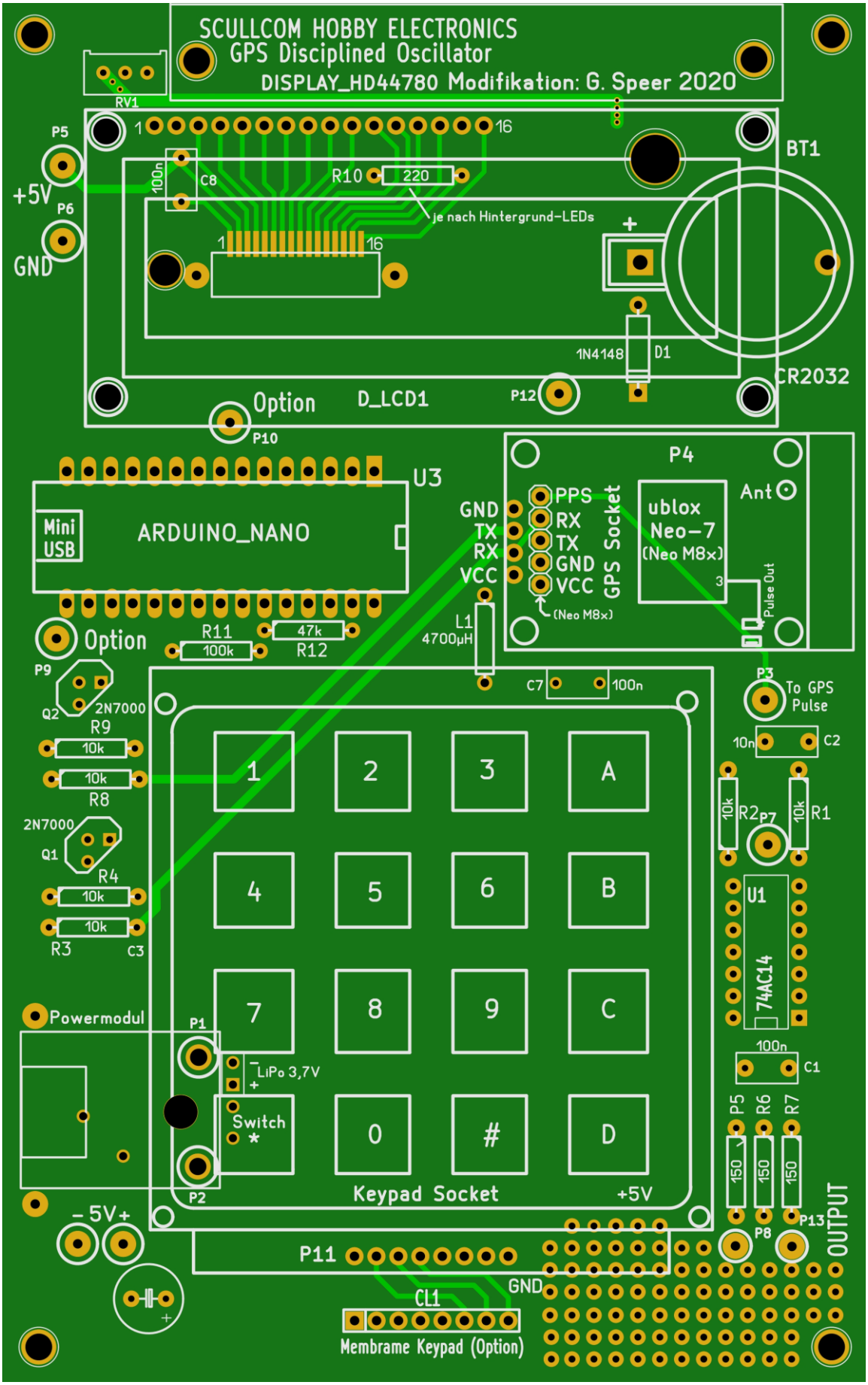
1. Weder das GPS-Modul noch das Arduino ziehen den Logikpegel auf LOW. Die Source des MOSFET und daher die serielle GPS-Leitung wird durch ihre 10K-Pull-up-Widerstände auf 3,3 V hochgezogen. Das Gate und die Source des MOSFET liegt bei 3,3 V, so dass sein VGS unter der Schwellenspannung liegt und der MOSFET daher nicht leitet.
Dadurch kann der Pegel am Arduino-Pin vom MOSFET-Drain-Pull-Up-Widerstand auf einen logischen Pegel von 5 V angehoben werden.
Die Leitungen beider Abschnitte sind also HIGH, jedoch auf einem anderen Spannungsniveau.

2. Der 3,3-V-Logikpegel des GPS-Moduls wird auf einen niedrigen Pegel heruntergezogen. Die Source des MOSFET wird ebenfalls LOW, während das Gate bei 3,3 V bleibt. VGS steigt über die Schwelle und der MOSFET beginnt zu leiten. Das Drain des MOSFET wird nun durch die Aktion des GPS-Moduls auf ein LOW Niveau heruntergezogen. Also jetzt beide, das GPS-Modul und der Arduino gehen auf den gleichen Spannungspegel.

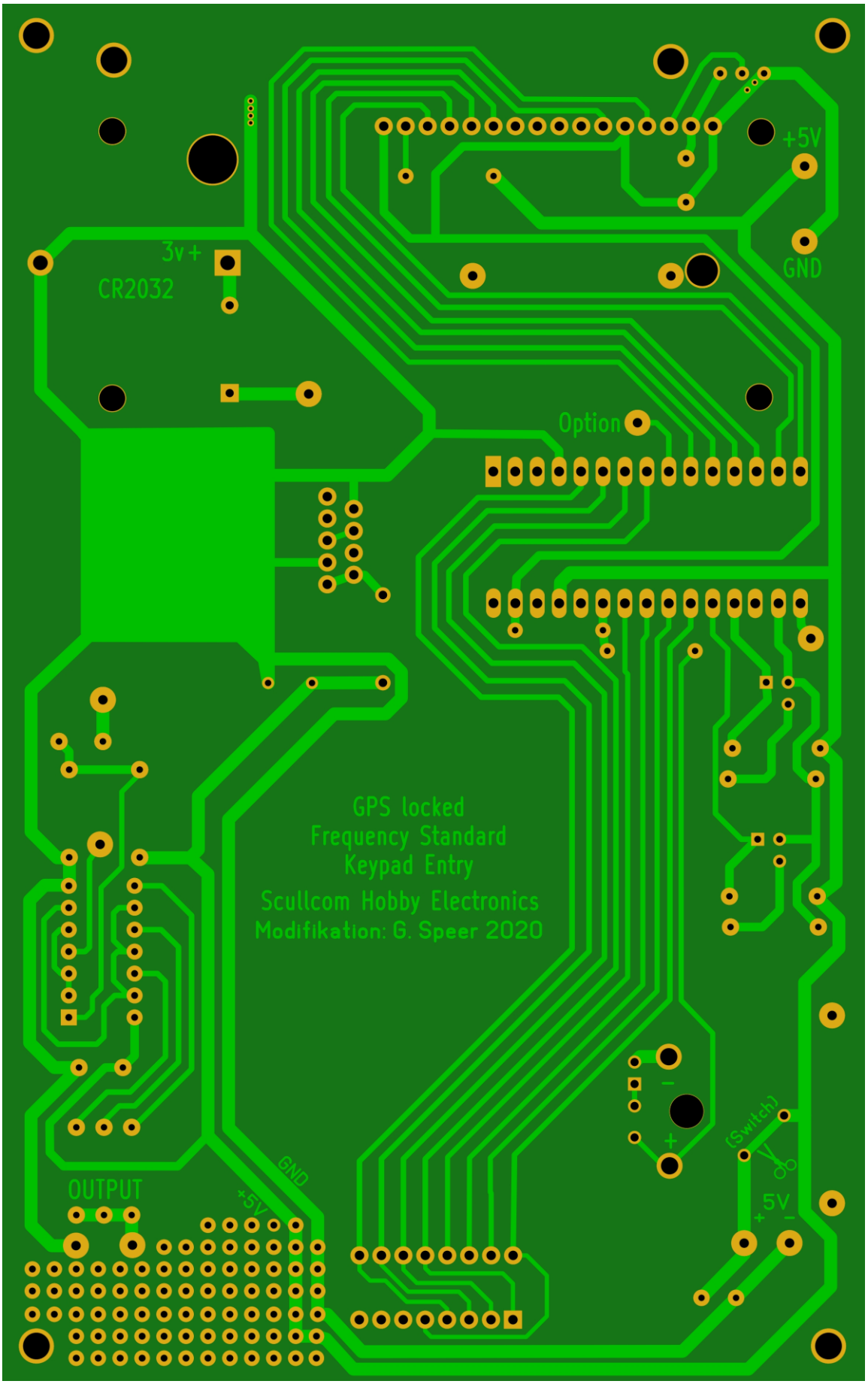
3. Der Arduino 5 V-Logikpegel am Drain des MOSFET wird vom Arduino auf einen LOW-Pegel heruntergezogen. Nun leitet und zieht die Drain-Substrat-Diode des MOSFET die Source des MOSFET bis VGS überschreitet die Schwelle und der MOSFET beginnt nach unten
zu leiten. Die Source des MOSFET ist dann weiter auf einen LOW-Pegel heruntergezogen,
durch diese Aktion des Arduino, der den 5-V-Logikpegel LOW annimmt. Also beide die
Arduino und das GPS-Modul gehen auf den gleichen Spannungspegel.

Die drei oben erläuterten Zustände zeigen, dass die Logikpegel unabhängig in beide Richtungen übertragen werden, unabhängig davon, welches Gerät steuert den Logikpegel.


GPS Locked Frequency Standard with Keypad entry
Schaltung geändert
To 9 Volt Battery (6 x AA batteries)
P2
GND P1
U2
C3 470uF
C4 100nF
C5 100nF
C6 100uF
+5V
P5
5 volt supply for option upgrades
GND
P6
+5V
DISPLAY_HD44780
D_LCD1
R10 220R
RV1 10K
C8 100nF
To GPS Module RAM Battery Back-up
P12
(Optional)
D1 1N4148
CR2032
BT1
3V Back-up Battery
R11 100K
10k
100nF
Future Option Upgrade Pins
P9
P10
U1B 74AC14
P7
Option
U1D 74AC14
R5 150
U1C 74AC14
U1E 74AC14
R6 150
P8
Frequency Output (50 ohm)
U1F 74AC14
R7 150
U1A 74AC14
P13
C1 100nF
+V
GND
4x4 Matrix Keypad
P11
Keypad Socket
ROWS
Membrame Keypad (Option)
7 8 9 A
4 5 6 B
1 2 3 C
* 0 # D
CL1
COLUMNS
U3
DIL30
ARDUINO_NANO
3V3
5V
Vin
GND
GND
RST
RST
+3.3V
Logic Level Converter
R1 10K
R2 10K
R3 10K
R4 10K
R8 10K
R9 10K
2N7000
Q1
2N7000
Q2
820nF
C2
P3
GPS Pulse Input
To GPS RX pin
L1 4700µH
C7 100nF
P4
GPS Socket
To GPS TX pin
GND
Sheet: /
File: 10MHz_GPS_Display.sch
Title: updated for double sided PCB 5th July 2018
Size: A4
Date: 22 mar 2016
KiCad E.D.A. kicad 4.0.7
Rev:
Id: 1/1


SCULLCOM HOBBY ELECTRONICS
GPS Disciplined Oscillator
DISPLAY_HD44780 Modifikation: G. Speer 2020
RV1
P5
+5V
P6
GND
C8
100n
R10
220
je nach Hintergrund-LEDs
BT1
CR2032
1N4148
D1
D_LCD1
Option
P10
P12
U3
Mini USB
ARDUINO_NANO
P4
GPS Socket
GND
TX
RX
VCC
PPS
RX
TX
GND
VCC
(Neo M8x)
ublox Neo-7 (Neo M8x)
Ant
Pulse Out
L1
4700µH
R11
100k
47k
R12
Option
P9
Q2
2N7000
R9
10k
10k
R8
2N7000
Q1
R4
10k
10k
R3
C3
C7
100n
P3
To GPS Pulse
10n
C2
10k
R2
P7
10k
R1
U1
74AC14
100n
C1
P5
R6
R7
150
150
150
P13
OUTPUT
P8
1
2
3
A
4
5
6
B
7
8
9
C
0
#
D
Keypad Socket
+5V
Powermodul
P1
LiPo 3,7V
Switch
*
P2
- 5V +
P11
GND
CL1
Membrame Keypad (Option)

Top


+5V
GND
3v+
CR2032
Option
GPS locked
Frequency Standard
Keypad Entry
Scullcom Hobby Electronics
Modifikation: G. Speer 2020
OUTPUT
+5V
GND
(Switch)
5V

Bottom

Lesen der GPS-Moduldaten, um die Anzahl der erkannten Satelliten zu ermitteln

Wenn das GPS-Modul mit dem Empfang von Satellitendaten beginnt, liefert es einen konstanten Strom bekannter Datensätze.
Innerhalb dieses Datenstroms umfassen die wichtigsten NMEA-Sätze die GGA, die bereitstellt die aktuellen Fix-Daten, das RMC, das die minimalen GPS-Satzinformationen bereitstellt, und das GSA, das die Informationen bereitstellt
die Satellitenstatusdaten. In unserem Projekt werden wir den GGA-Datenstrom verwenden, um die Anzahl der erkannten Satelliten zu extrahieren
und verriegelt an.
Unten finden Sie ein Beispiel für einen typischen GGA-Datensatz:
$ GPGGA, 123519.00.4807.03830, N, 01131.00000, E, 1.08.0.9.545.4, M, 46.9, M ,, * 47
Wo:
Fixdaten des GGA Global Positioning System
123519.00 Fix genommen um 12:35:19 UTC (aktuelle Zeit)
4807.03830, N Breite 48 Grad 07.038 'N.
01131.00000, E Länge 11 Grad 31.000 'E.
1 Fixqualität: 0 = ungültig, 1 = GPS-Fix (SPS), 2 = DGPS-Fix, 3 = PPS-Fix, 4 = Echtzeitkinematik,
5 = Float RTK, 6 = geschätzt (Dead Reckoning) (2.3 Feature), 7 = manueller Eingabemodus,
8 = Simulationsmodus
08 Anzahl der verfolgten Satelliten
0,9 Horizontale Positionsverdünnung
545,4, M Höhe, Meter, über dem mittleren Meeresspiegel
46,9, M Höhe des Geoids (mittlerer Meeresspiegel) über dem WGS84-Ellipsoid
(leeres Feld) Zeit in Sekunden seit dem letzten DGPS-Update
(leeres Feld) DGPS-Stations-ID-Nummer
* 47 Die Prüfsummenangaben beginnen immer mit *
Wir werden die oben lila hervorgehobenen Daten verwenden, um die Anzahl der Satelliten auf unserem LCD anzuzeigen. In Ordnung
Um diese Informationen zu extrahieren, richten wir einen zweiten seriellen Software-Port ein.
Zuerst müssen wir die NMEA-Daten identifizieren, die wir erfassen müssen, und dies tun wir mit dem fließenden Code:

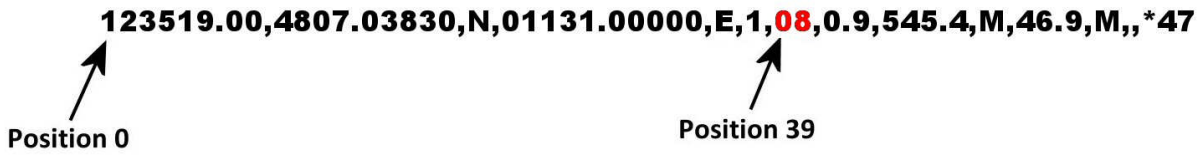
**char gpsin[8] = "$GPGGA,";** //define a character string “$GPGGA,”
if (GPSreceiver.find(gpsin)) { //if serial data starts with the characters “$GPGGA,”
GPGGAdata = GPSreceiver.readStringUntil('\n'); //read data until carriage return

Wenn die Zeichen "$ GPGGA" erkannt werden, beginnt der Rest der Zeichenfolgendaten in das zu lesen
Puffer. Die erfassten Daten beginnen also mit der ersten Ziffer der 'Zeit'-Daten (und nicht mit dem $ - Zeichen). Das
Die erfassten Daten sehen dann wie folgt aus:

123519.00,4807.03830,N,01131.00000,E,1,08,0.9,545.4,M,46.9,M,,*47

In der Hauptschleife verwenden wir den Befehl substring, um die Anzahl der Satelliten aus dem GGA zu extrahieren
Datensatz. Dies ist eine einfache und einfache Möglichkeit, die Daten zu extrahieren, die Sie verwenden möchten. Der Teilstring
Befehl ist wie folgt definiert:
Teilzeichenfolge (Index, bis) - Gibt eine Zeichenfolge zurück, deren Zeichen vom Index bis zum Ende beginnen

Zeichenposition vor dem zu
So extrahieren Sie die Anzahl der Satelliten (hervorgehoben beim Lesen) aus dem folgenden GGA-Datensatz



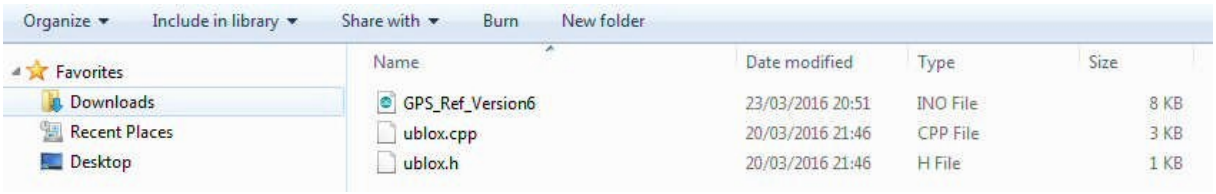
Wir verwenden den folgenden Code:
sats = GPGGAdata.substring (39,41);
Die Zeichenfolge "sats" ist jetzt gleich "08"
Wenn Sie andere Daten extrahieren möchten, müssen Sie lediglich die Indexnummern in der Teilzeichenfolge ändern.
Um beispielsweise die aktuelle Zeit in eine mit Zeichenfolgen beschriftete Zeit zu extrahieren, würden Sie Folgendes verwenden:
time = GPGGAdata.substring(7,13); //the string “time” now is equal to “123519”

Details des Codes, der in der Programmschleife verwendet wird, um die Anzahl der Satelliten zu extrahieren und auf dem LCD anzuzeigen

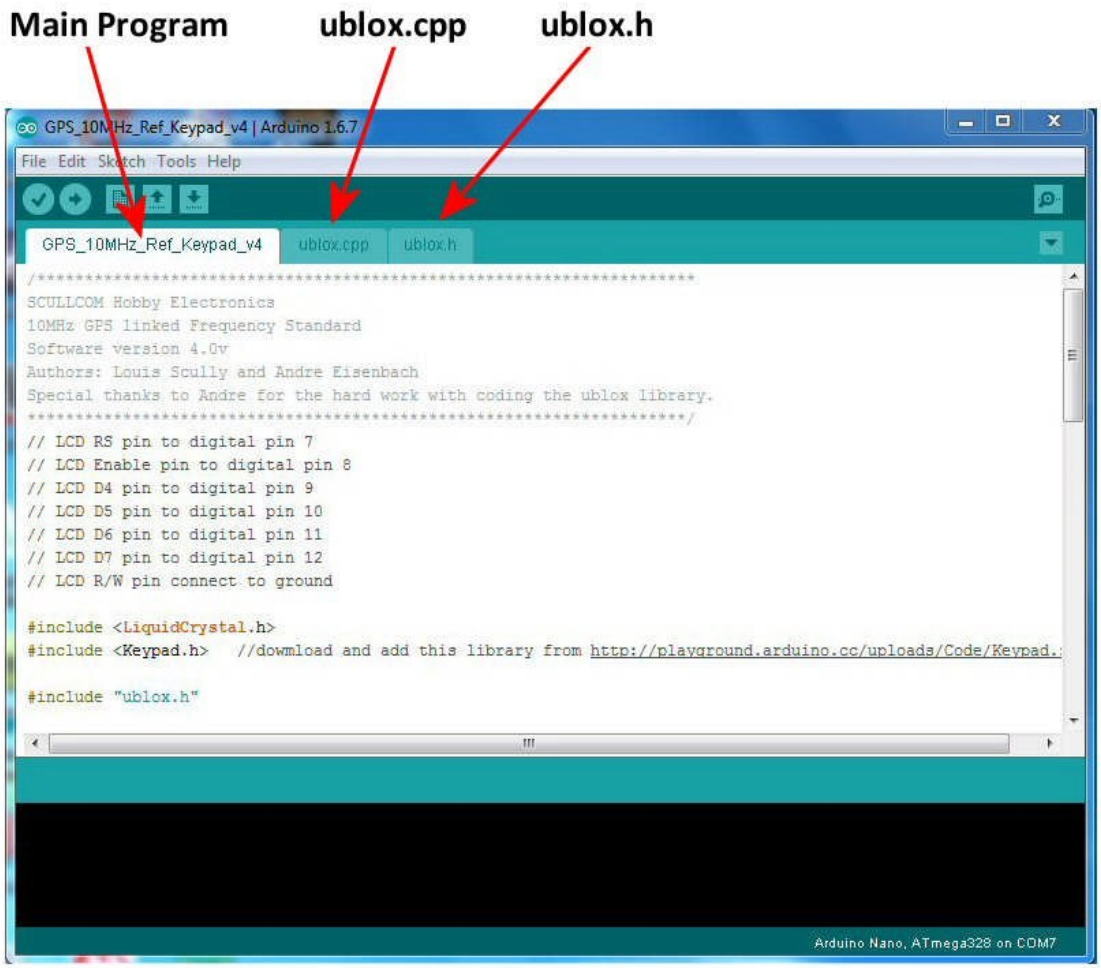
```
//new variables added at start of program
String sats; //number of satellites detected
String GPGGAdata; //satellte data string from GPS module
String FreqStatus; //used to print Frequency locked if applicable
int satNum; //number of satellites detected as an interger
//set up a Software Serial port called “GPSreceiver”
SoftwareSerial GPSreceiver (14,255); //14 (A0) is RX in on Arduino connected to GPS TX pin.
//255 a nonexsistent pin number used to free up a pin
//in void setup() include the serial monitor command below and set serial monitor to 57600 baud
Serial.begin(57600); //this is simply used for testing
//The # key will be used to tell the unit to display the number of satellites on the LCD.
// below is the software code used in the main loop to show number of satellites when “#” key pressed:
case '#': //# pressed Show number of satellites detected and frequency status
lcd.clear(); //clear LCD
lcd.print("Satellites ="); //print the word “Satellites =” on the top line of the LCD
do { //now loop
GPSreceiver.flush(); //clear any data which may be in the Serial Buffer
if (GPSreceiver.available()) //if any GPS data is available
{
GPSreceiver.read(); //read GPS serial data streams if available
}
char gpsin[8] = "$GPGGA,"; //define a character string “$GPGGA,”
if (GPSreceiver.find(gpsin)) { //if serial data starts with the characters “$GPGGA,”
GPGGAdata = GPSreceiver.readStringUntil('\n'); //read data until a carriage return
sats = GPGGAdata.substring(39,41); //extract number of satellites from data string
}
satNum = sats.toInt(); //convert sats string to an integeter so as to remove leading zero
if (satNum >0){
FreqStatus = "Frequency Locked"; //if satellites detected the status string = “Frequency Locked”
}else{
FreqStatus = " "; //16 spaces to clear line if no lock
}
Serial.print(GPGGAdata); //These serial print lines are for testing using Serial Monitor
Serial.println(); //full GPGGA data sentence printed and then carriage return
Serial.print("Satellite = ");
Serial.print(satNum);
Serial.println();
Serial.print(FreqStatus);
Serial.println();
lcd.setCursor(13,0); //set cusor position to column 13 of row 0 (first line on LCD)
lcd.print(" "); //clears number if no satellites detected (3 spaces used)
lcd.setCursor(13,0); //set cusor position to column 13 of row 0 (first line on LCD)
lcd.print(satNum); //print number of satellites to LCD
lcd.setCursor(0,1); //set cusor position to column 0 of row 1 (second line on LCD)
lcd.print(FreqStatus); //print Frequency Locked status
}
while (customKeypad.getKey()!= '#'); //if # key pressed again break out of loop
displayCurrentFrequency(); //and display current frequency subroutine
break; //break and return to main program loop
```

Die Software für dieses Projekt besteht aus drei Teilen, die alle in einer Download-Zip-Datei enthalten sind.

Sobald die Datei entpackt ist, haben Sie einen Ordner wie oben. In diesem Ordner finden Sie die folgenden angezeigten Dateien
unten:



Stellen Sie sicher, dass Sie die neueste Version der Arduino IDE ausführen, nämlich "Arduino 1.6.7" (ältere Versionen)
funktioniert möglicherweise nicht). Führen Sie die Datei "GPS_Ref_Version6" aus und Sie sollten feststellen, dass die Software in der Arduino IDE als geöffnet ist
unten gezeigt:



Kompilieren Sie nun die Software und laden Sie sie auf den Arduino nano auf der Hauptplatine hoch.

```
/*****************************************************************
SCULLCOM Hobby Electronics
10MHz GPS linked Frequency Standard
Software version 6.0v
24th March 2016
Keypad frequency entry and Satellite data
Authors: Louis Scully and Andre Eisenbach
Special thanks to Andre for the hard work with coding the ublox library.
*****************************************************************/
// LCD RS pin to digital pin 7
// LCD Enable pin to digital pin 8
// LCD D4 pin to digital pin 9
// LCD D5 pin to digital pin 10
// LCD D6 pin to digital pin 11
// LCD D7 pin to digital pin 12
// LCD R/W pin connect to ground
#include <LiquidCrystal.h> //LCD library
#include <Keypad.h> //dowmload and add this library from
http://playground.arduino.cc/uploads/Code/Keypad.zip
#include "ublox.h"
#define PIN_GPS_RX 255 //was previously 14 changed to 255 a nonexsistent pin number used to free
up a pin
#define PIN_GPS_TX 15
UBloxGPS gps(PIN_GPS_RX, PIN_GPS_TX); //sets up ublox GPS Software Serial port
SoftwareSerial GPSreceiver (14,255); //14 (A0) is RX in on Arduino connected to GPS TX pin.
//255 a nonexsistent pin number used to free up a pin
LiquidCrystal lcd(7, 8, 9, 10, 11, 12); // (RS, E, D4, D5, D6, D7)
long StartFreq = 1000000; //sets initial frequency of locked GPS frequency at 1MHz
long SetFreq = 0;
float NewFreq = StartFreq;
float CurrentFrequency = 0;
const float R11 = 100000; //Resistor values in ohms for the volt meter divider
const float R12 = 47000;
const float RefVolt = 5; //use default reference of 5 volt on Arduino
float ResistorDivideFactor = 0;
float voltage_reading = 0;
float battery_voltage = 0;
String sats; //number of satellites detected
String GPGGAdata; //satellte data string from GPS module
String FreqStatus; //used to print Frequency locked if applicable
int satNum; //number of satellites detected as an interger
String range; //frequency range for display either MHz, KHz or Hz
int decimalPlaces; //number of decimal places used dependant on range
char customKey; //key pressed
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
```

```
{'*','0','#','D'}
};
byte rowPins[ROWS] = {5, 4, 3, 2}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {16, 17, 18, 19}; //connect to the column pinouts of the keypad
//initialize an instance of class NewKeypad
Keypad customKeypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS);
void setup() {
Serial.begin(57600);
GPSreceiver.begin(9600);
lcd.begin(16, 2);
lcd.print("SCULLCOM");
lcd.setCursor(0,1);
lcd.print("Freq Standard v6");
delay(3000);
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Frequency");
gps.writeFrequency(StartFreq); //sets start up frequency when locked
lcd.setCursor(0,1);
lcd.print(StartFreq);
lcd.setCursor(14,1);
lcd.print("Hz");
}
void loop() {
customKey = customKeypad.getKey();
switch(customKey)
{
case '0' ... '9': // accept number input
lcd.setCursor(0,0);
SetFreq = SetFreq * 10 + (customKey - '0');
lcd.setCursor(0,1);
lcd.print(SetFreq);
break;
case 'A': //enter new frequency in Hertz
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Set Frequency");
lcd.setCursor(0,1);
break;
case 'B': //send new frequency to GPS module
NewFreq=SetFreq;
gps.writeFrequency(NewFreq);
delay(1000);
displayCurrentFrequency();
SetFreq = 0;
break;
case 'C': //battery voltage check
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Battery Check");
voltage_reading = analogRead(A6);
ResistorDivideFactor = 1023 * (R12/(R11+R12));
battery_voltage = (voltage_reading * RefVolt)/ResistorDivideFactor;
```

```
lcd.setCursor(0,1);
lcd.print(battery_voltage,2);
lcd.print(" Volts");
break;
case 'D': //display current frequency
displayCurrentFrequency();
break;
case '*': //For future upgrade
break;
case '#': //Show number of satellites detected and frequency locked status
lcd.clear();
lcd.print("Satellites =");
do {
GPSreceiver.flush();
if (GPSreceiver.available())
{
GPSreceiver.read();
}
char gpsin[8] = "$GPGGA,";
if (GPSreceiver.find(gpsin)) {
GPGGAdata = GPSreceiver.readStringUntil('\n');
sats = GPGGAdata.substring(39,41); //extract number of satellites data
}
satNum = sats.toInt(); //convert sats string to an integeter so as to remove leading zero
if (satNum >0){
FreqStatus = "Frequency Locked";
}else{
FreqStatus = " "; //16 spaces to clear line if no lock
}
Serial.print(GPGGAdata); //These serial print lines are for testing using Serial Monitor
Serial.println(); //full GPGGA data sentence and then carriage return
Serial.print("Satellite = ");
Serial.print(satNum);
Serial.println();
Serial.print(FreqStatus);
Serial.println();
lcd.setCursor(13,0); //set cusor position to column 13 of row 0 (first line on LCD)
lcd.print(" "); //clears number if no satellites detected (3 spaces used)
lcd.setCursor(13,0); //set cusor position to column 13 of row 0 (first line on LCD)
lcd.print(satNum); //print number of satellites to LCD
lcd.setCursor(0,1); //set cusor position to column 0 of row 1 (second line on LCD)
lcd.print(FreqStatus); //print Frequency Locked status
}
while (customKeypad.getKey()!= '#'); //if # key pressed again break out of loop
displayCurrentFrequency(); //and display current frequency
break; //break and return to main program loop
}
}
void displayCurrentFrequency(){ //subroutine to display current frequency
lcd.clear();
CurrentFrequency = NewFreq; //current frequency is set to new frequency entered
if (CurrentFrequency >=1000000)
{
```

```
CurrentFrequency = CurrentFrequency / 1000000;
range = "MHz";
decimalPlaces = 6; //sets number of decimal places to 6 if MHz range used
}
else if (CurrentFrequency >=1000)
{
CurrentFrequency = CurrentFrequency /1000;
range = "KHz";
decimalPlaces = 3; //sets number of decimal places to 3 if KHz range used
}
else
{
range = "Hz";
decimalPlaces =0; //sets number of decimal places to 0 if Hz range used
}
lcd.print("Frequency");
lcd.setCursor(0,1);
lcd.print(CurrentFrequency,decimalPlaces);
lcd.setCursor(13,1);
lcd.print(range);
}
/**************************************************************************
ublox GPS library
Created by Andre Eisenbach for SCULLCOM Hobby Electronics
**************************************************************************/
#include "ublox.h"
#include <stdint.h>
#include <string.h> // formemset()
namespace
{
struct GPS_TP5_MSG
{
uint8_t header1;
uint8_t header2;
uint8_t message_class;
uint8_t message_id;
uint16_t length;
uint8_t timepulse_idx;
uint8_t version;
uint16_t reserved;
int16_t antenna_cable_delay;
int16_t rf_group_delay;
uint32_t frequency_unlocked;
uint32_t frequency_locked;
uint32_t duty_cycle_unlocked;
uint32_t duty_cycle_locked;
uint32_t user_delay;
uint32_t flags;
uint16_t checksum;
} __attribute__((packed));
const uint16_t GPS_TP5_ACTIVE = 0x01;
const uint16_t GPS_TP5_SYNC_TO_GNSS = 0x02;
const uint16_t GPS_TP5_LOCKED_OTHER_SET = 0x04;
const uint16_t GPS_TP5_IS_FREQUENCY = 0x08;
```

```
const uint16_t GPS_TP5_IS_LENGTH = 0x10;
const uint16_t GPS_TP5_ALIGN_TO_TOW = 0x20;
const uint16_t GPS_TP5_POLARITY_RISING = 0x40;
const uint32_t GPS_TP5_DUTY_CYCLE_50 = 0x80000000;
uint16_t calculateChecksum(uint8_t *p, size_t len)
{
uint8_t ck_a = 0;
uint8_t ck_b = 0;
while (len--)
{
ck_a = ck_a + *p++;
ck_b = ck_b + ck_a;
}
return ck_a | (ck_b << 8);
}
}
UBloxGPS::UBloxGPS(uint8_t pin_rx, uint8_t pin_tx)
: ss_(SoftwareSerial(pin_rx, pin_tx))
{
ss_.begin(9600);
}
void UBloxGPS::writeFrequency(uint32_t freq)
{
GPS_TP5_MSG msg;
memset(&msg, 0, sizeof(msg));
msg.header1 = 0xB5;
msg.header2 = 0x62;
msg.message_class = 0x06;
msg.message_id = 0x31;
msg.length = 32;
msg.version = 1;
msg.antenna_cable_delay = 50;
msg.frequency_unlocked = 3333; //this sets the unlocked frequency at 3333Hz (may be
changed)
msg.duty_cycle_unlocked = GPS_TP5_DUTY_CYCLE_50;
msg.duty_cycle_locked = GPS_TP5_DUTY_CYCLE_50;
msg.flags = GPS_TP5_ACTIVE | GPS_TP5_SYNC_TO_GNSS | GPS_TP5_LOCKED_OTHER_SET |
GPS_TP5_IS_FREQUENCY;
msg.frequency_locked = freq;
msg.checksum = calculateChecksum(((uint8_t*)&msg) + 2, 4 +msg.length);
ss_.write((char*)&msg, sizeof(msg));
}
/*********************************************************************
ublox GPS library
Created by Andre Eisenbach for SCULLCOM Hobby Electronics
*********************************************************************/
#ifndef __UBLOX_H
#define __UBLOX_H
#include <SoftwareSerial.h>
class UBloxGPS
{
public:
UBloxGPS(uint8_t pin_rx, uint8_t pin_tx);
void writeFrequency(uint32_t freq);
private:
```

```
SoftwareSerial ss_;
};
#endif // __UBLOX_H
```