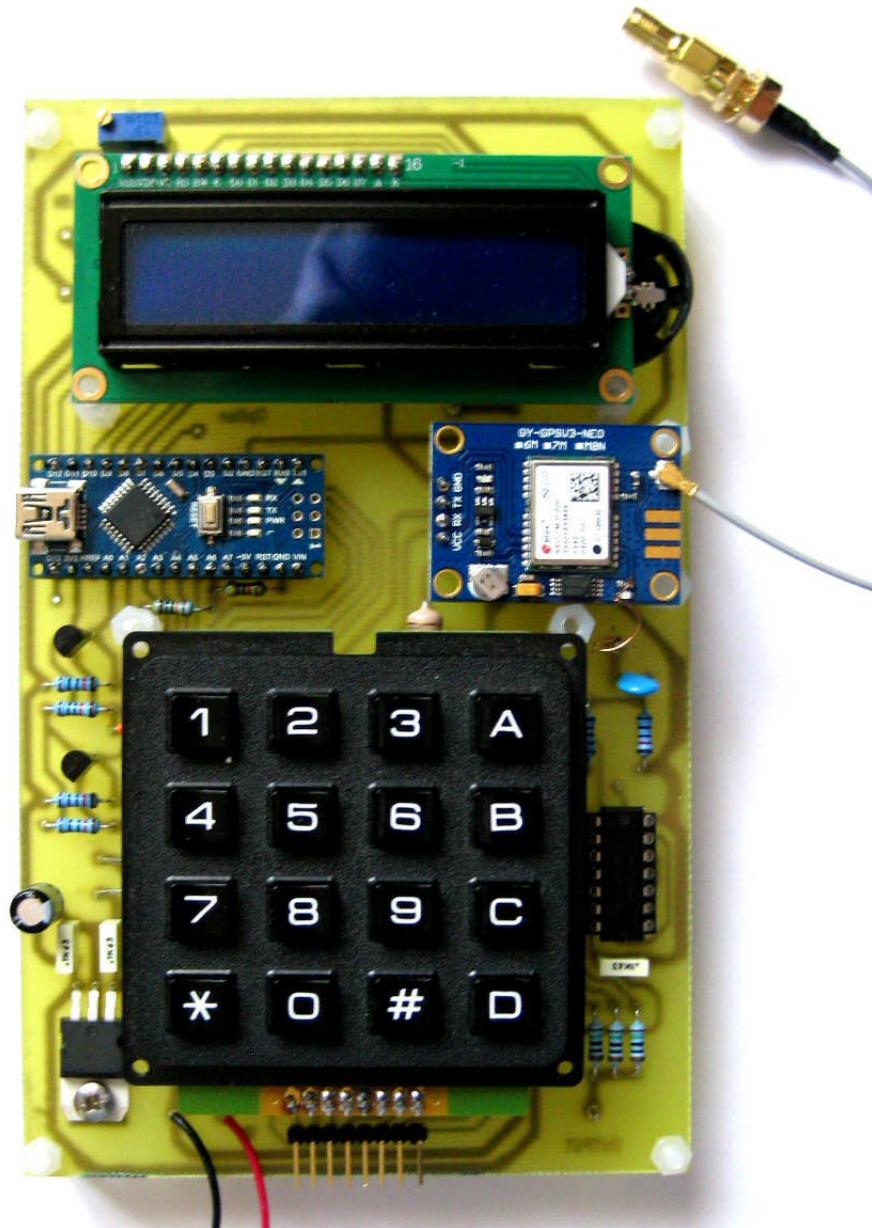


SCULLCOM

Hobby Electronics



GPS Locked

Frequency Reference Standard

(Keypad Version)

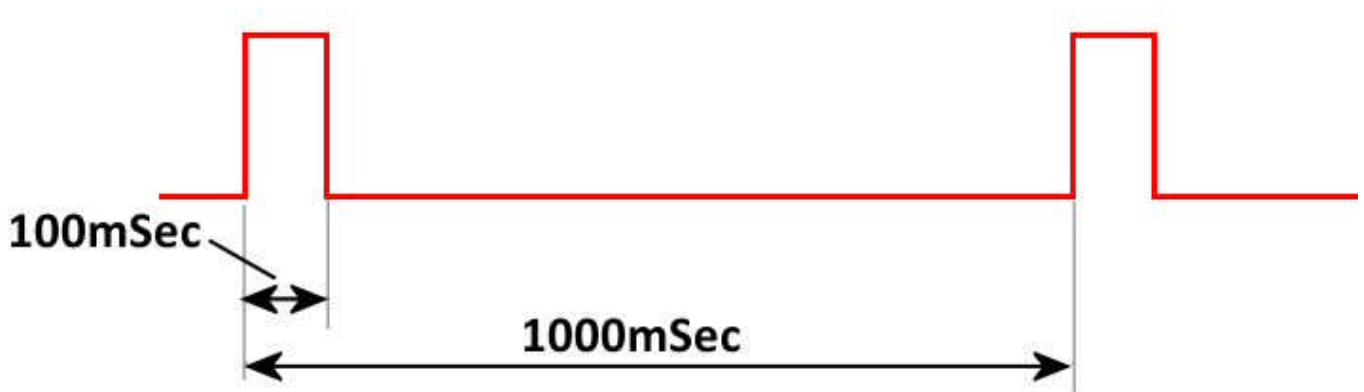
Construction Manual (updated)

GPS Locked Frequency Standard Project

This project will use a small GPS module which has the u-blox Neo-7 GPS receiver. These modules are readily available on sites like eBay for around £12, they tend to be advertised for use with flight controllers on model aircraft.

The u-blox receiver is normally fitted in the centre of a small PCB. Additional components on the PCB include a 3.3 volt Low Drop Voltage Regulator, timepulse and power on LED's and EEPROM. There is also some short term back-up supply for the DRAM of the Neo-7 receiver by use of a 0.08F super capacitor (Seiko XH414HG). These capacitors normally only provide power off back-up for around a day or so if you are lucky.

These GPS modules are usually configured by default to provide a 1Hz timing pulse output which is normally used to flash a green LED to indicate when the unit has a fix on satellites and the local oscillator is locked to the GPS signal. By default this timing pulse provides a 100mSec pulse every 1000mSec, as shown below.



This timing pulse can be configured to provide different frequencies and different duty cycle and we will be using this option in this project to change the frequency and fix the duty cycle to 50%.

GPS satellites are primarily used as a navigation system; this global positioning system can also be used to disseminate precise time, time intervals, and frequency. The GPS carrier signals originate from an on-board atomic clock (oscillator) which is also monitored and controlled by ground stations in the United States to ensure it agrees with the Coordinated Universal Time (UTC). UTC is the primary time standard by which the world regulates clocks and time.

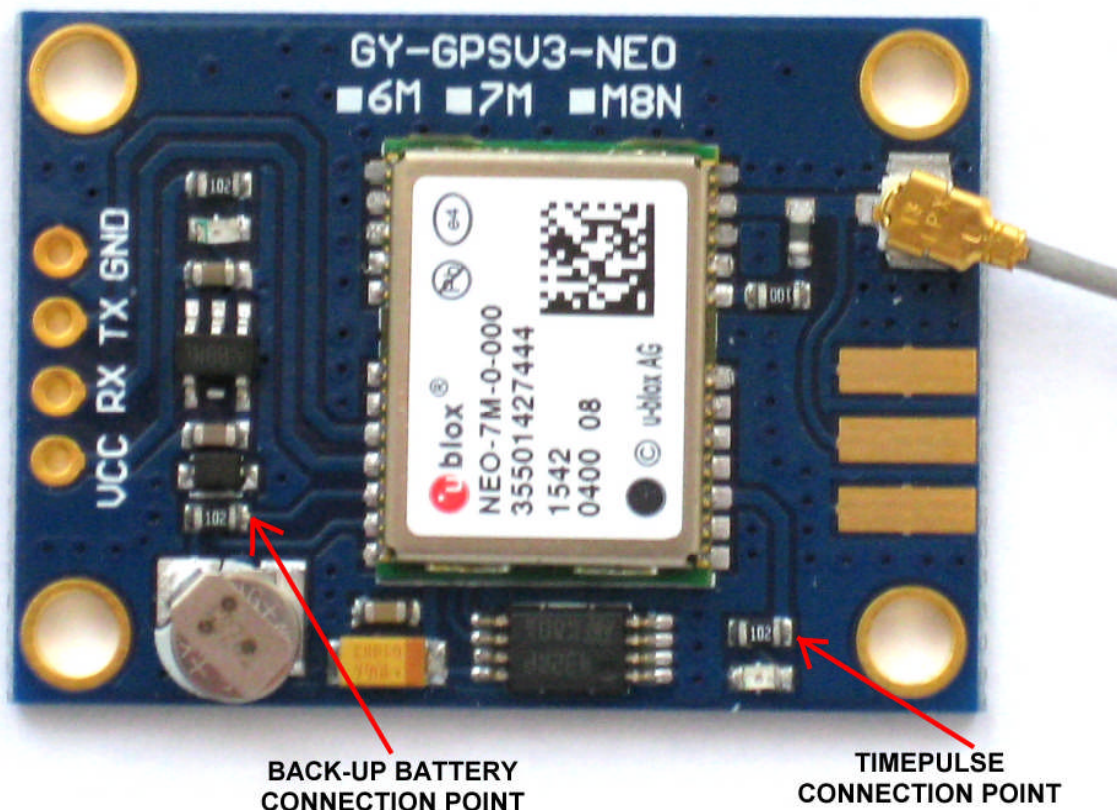
Atomic clocks are based on the natural atomic oscillations of gases in resonant cavities. When isolated from magnetic fields, rubidium and cesium gases will resonate at specific frequencies under controlled conditions. These frequencies are so accurate that since 1967 the length of the second has been defined as the frequency of a specific resonant mode of the cesium atom, producing 9,192,631,770 oscillations in one second.

The u-blox receiver uses a built-in 48MHz oscillator.

There are various versions of the u-blox receiver available, some of which can be configured to provide a frequency pulse output in the range 0.25Hz to 10MHz. Those which are suitable for this project are listed in the table below:

u-blox Receiver Type	Type of Memory fitted	Type of Oscillator used
Neo-6T	ROM	TCXO
Neo-7M	ROM	Crystal
Neo-7N	FLASH	TCXO
Neo-7P	FLASH	Crystal
Neo-M8M	ROM	Crystal
Neo-M8N	FLASH	TCXO
Neo-M8Q	ROM	Crystal

In this project we will use the Neo-7M. As this uses a ROM, the option to use a 3 volt backup battery on the PCB will help to speed up satellite lock after switch-on (this is optional). If the back-up battery is not used it may take a few minutes before the unit locks on to satellites. If the Neo-7N version is used then as it uses a FLASH memory there is no need for a back-up supply.

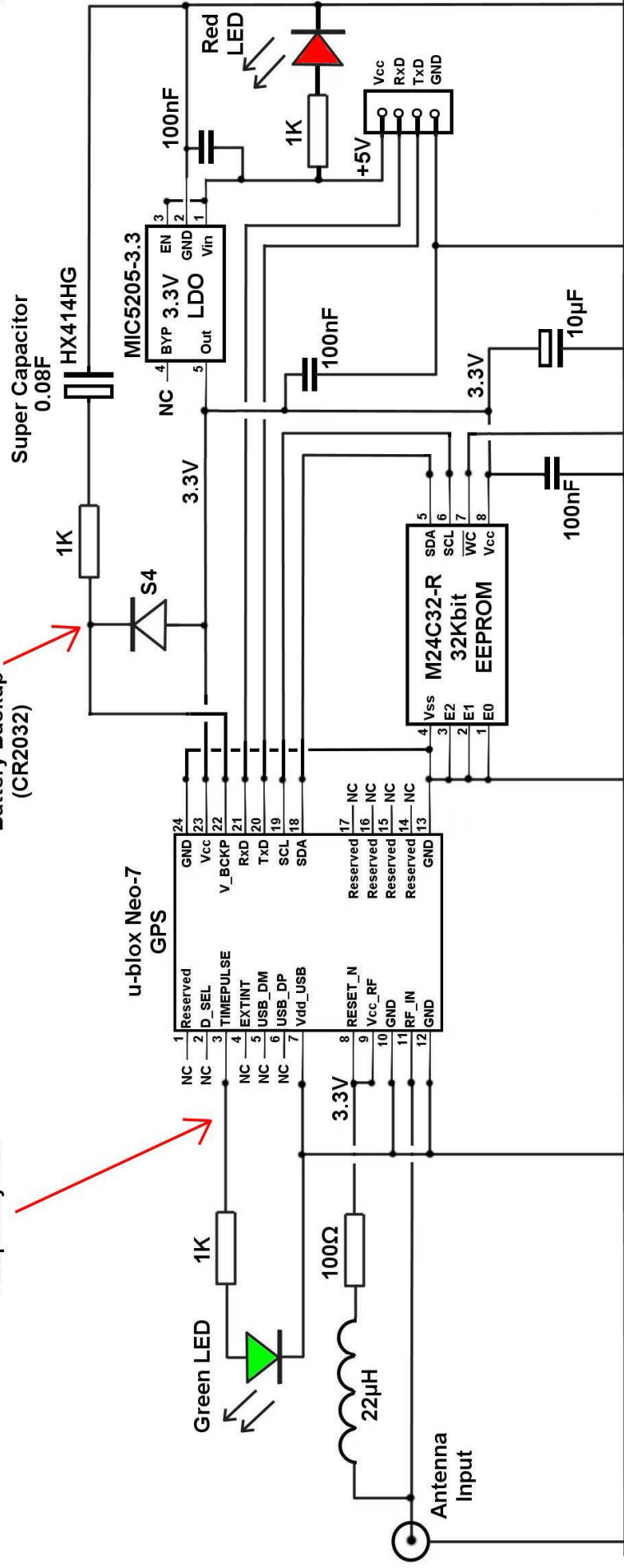


Connection points required. Solder on some connection wire. Use low gauge wire so as not to put too much stress on the module PCB tracks.

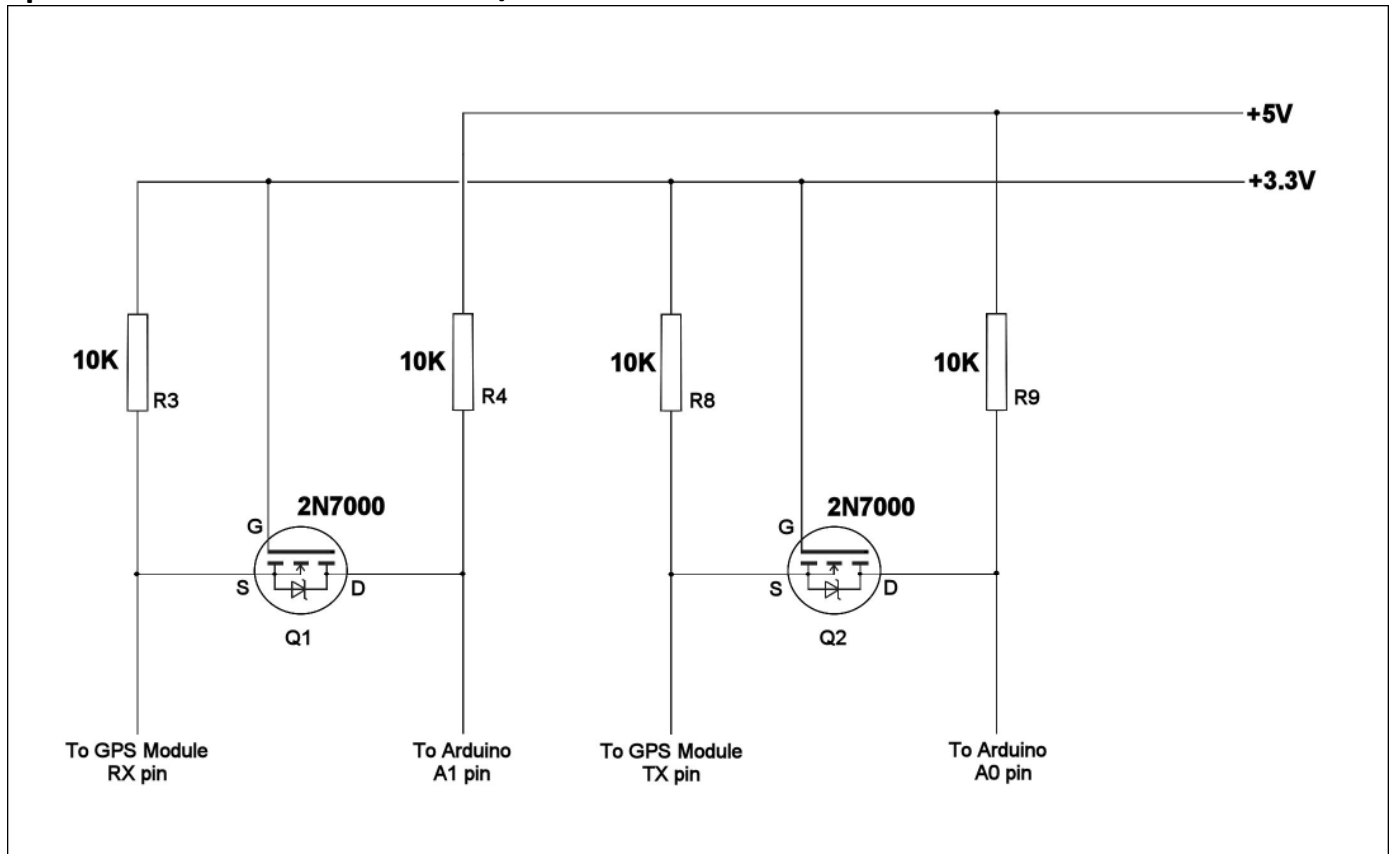
Connection Point
for Timepulse
Frequency Out

Connection point
for DRAM
Battery Backup
(CR2032)

u-blox Neo-7
GPS



Operation of the level converter/shifter circuit used



The u-blox Neo-7 GPS module operates at 3.3 volts and therefore care should be taken when connecting GPS serial TX and RX pins directly to the Arduino which is operating at 5 volt. Although in practice if only the GPS TX serial pin is used it seems to work fine as the Arduino appears to be able to handle lower logic level inputs from the GPS module. However, if the GPS RX pin is also connected to the Arduino this may cause damage to the GPS u-blox Neo-7 as the Arduino will drive it with 5 volt logic levels. Therefore, it is technically correct to use a logic level converter circuit to shift the 5 volt logic levels to 3 volt logic levels.

The above circuit which uses two 'N' channel MOSFETS provides a simple method of converting/shifter these logic levels and we use this technique in our project.

The logic level converter idea has been around for some time. During the late 1990's Philips R & D Engineers in The Netherlands, had some issues with logic level problems with their I²C Bus technology and there engineers came up with this idea of using a 'N' Channel MOSFET to shift the logic high levels. At the time developments in IC technology was improving which resulted in the clearances on the silicon chip substrate reducing to less than 0.5 μ m which limited the logic high level and supply voltages to around 3.3 volt.

This bi-directional level converter/shifter is used to connect the serial TX and RX pins of the GPS module to the Arduino, each with a different supply voltage and different logic levels. The above circuit has two logic level shifters, both identical. So we will consider the left hand side of the diagram. The source of the MOSFET, which is connected to the GPS module, has pull-up resistors connected to a 3.3 Volt supply, whilst the drain of the MOSFET, which is connected to the Arduino, has pull-up resistors connected to a 5 Volt supply.

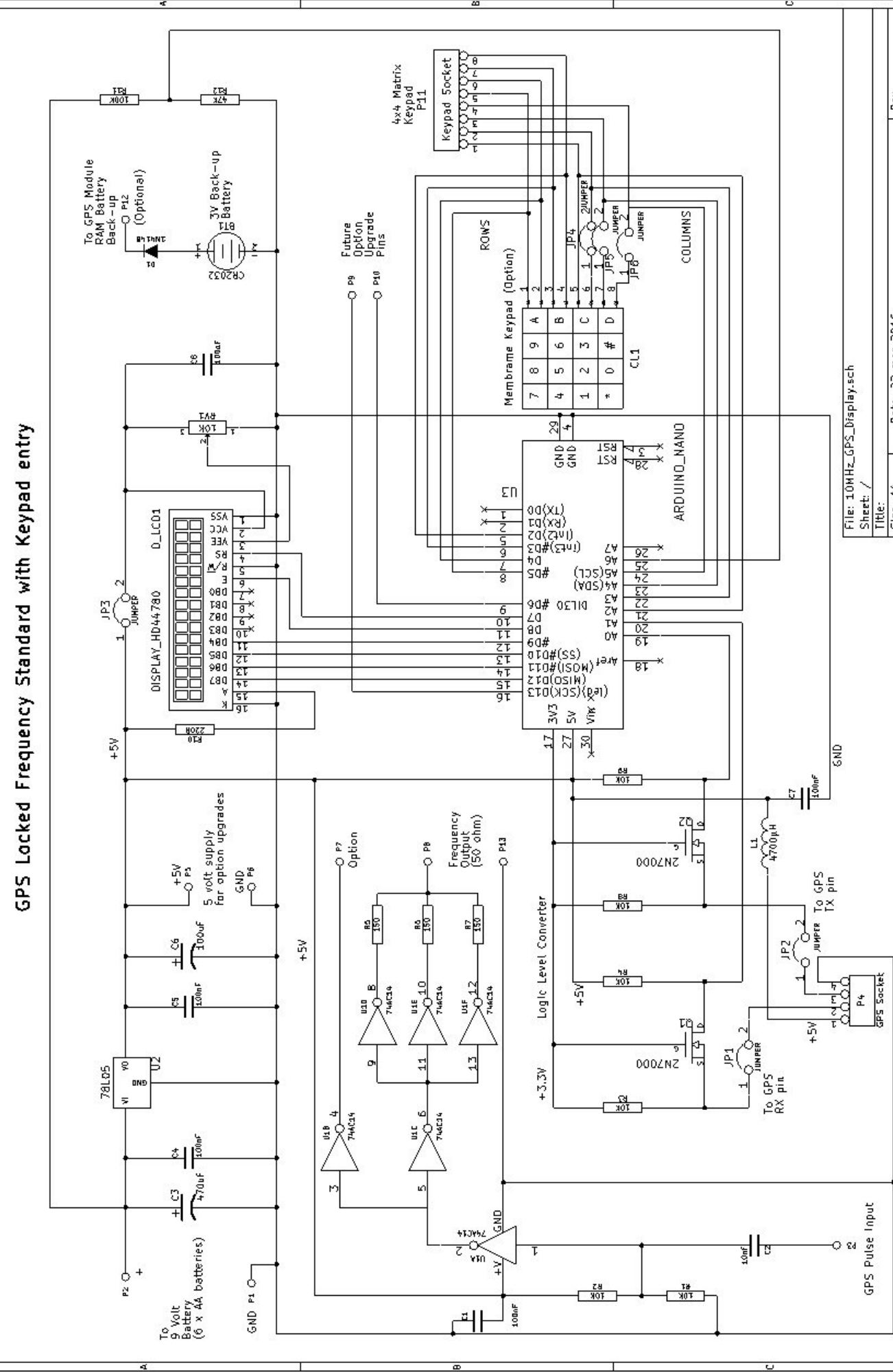
The following three states should be considered during the operation of this level converter/shifter:

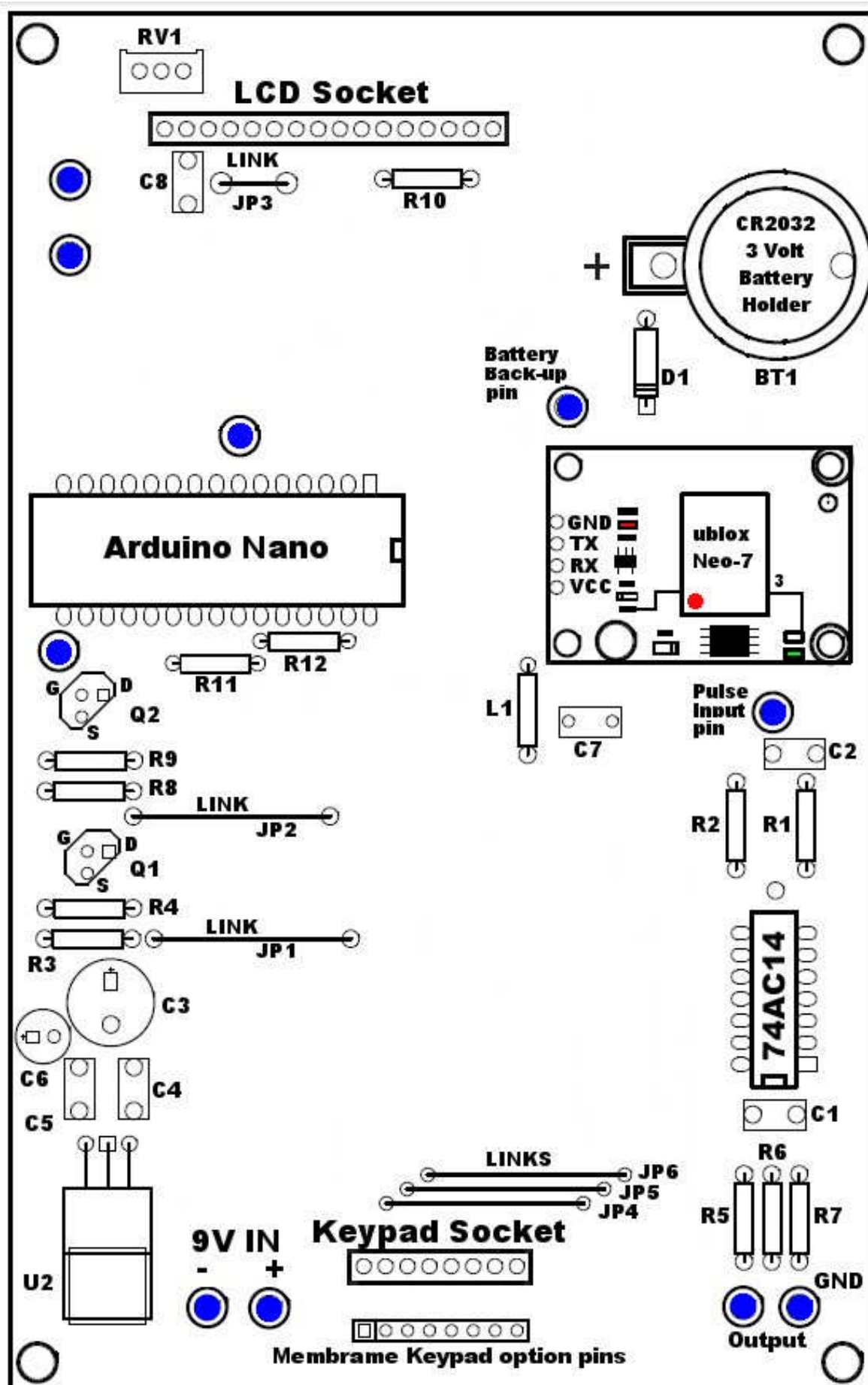
1. Neither the GPS module nor the Arduino is pulling the logic level LOW. The source of the MOSFET and therefore the GPS serial line is pulled up by its 10K pull-up resistors to 3.3 V. The gate and the source of the MOSFET is at 3.3 V, so its VGS is below the threshold voltage and the MOSFET is therefore not conducting. This allows the level at Arduino pin to be pulled up by the MOSFET drain pull-up resistor to 5 V logic level. So the lines of both sections are HIGH, but at a different voltage level.

2. The GPS module 3.3 V logic level is pulled down to a LOW level. The source of the MOSFET also becomes LOW, while the gate stays at 3.3 V. VGS rises above the threshold and the MOSFET starts to conduct. The drain of the MOSFET is now pulled down to a LOW level by the action of the GPS module. So now both the GPS module and the Arduino go LOW to the same voltage level.

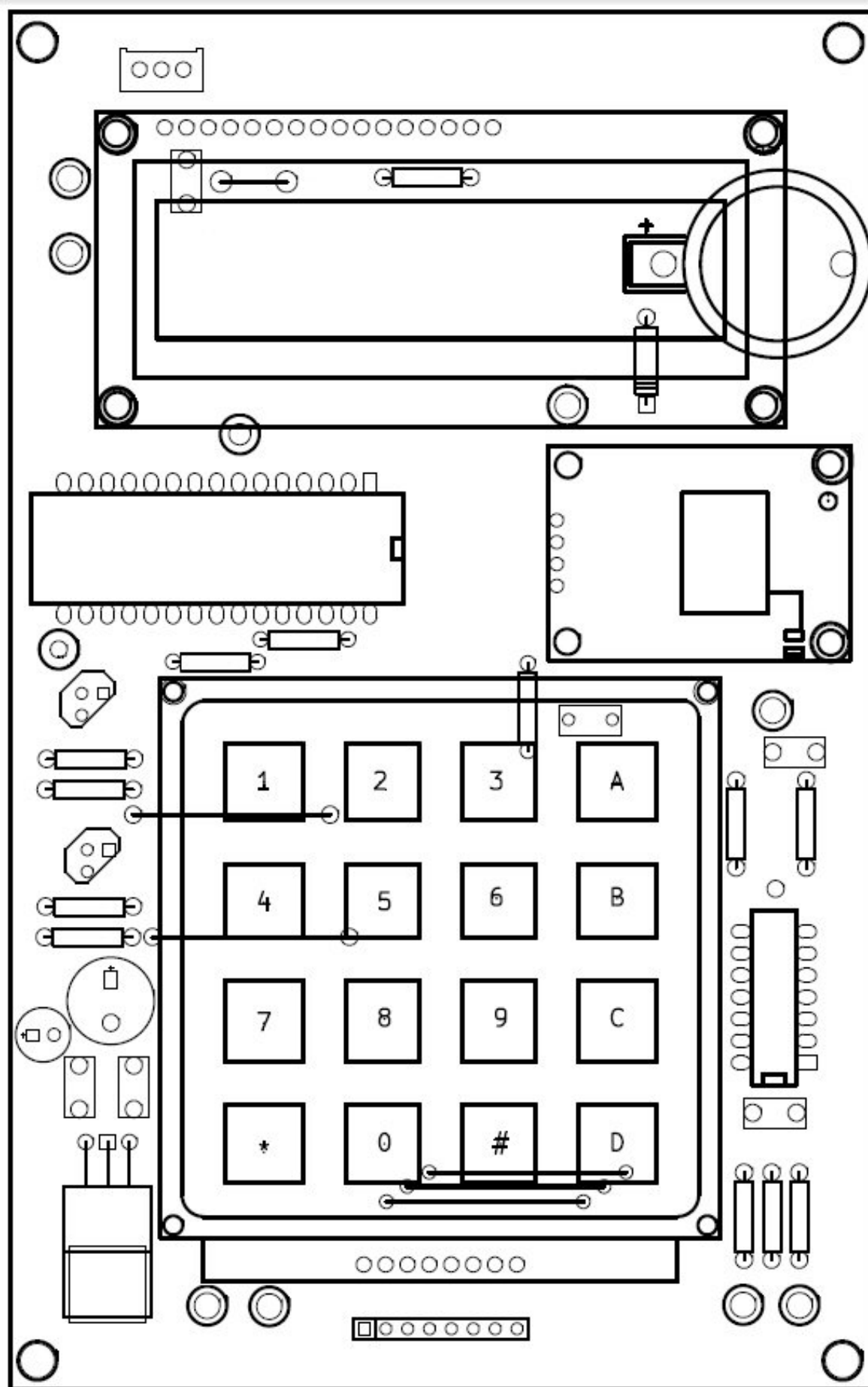
3. The Arduino 5 V logic level at the drain of the MOSFET is pulled down to a LOW level by the Arduino. Now the drain-substrate diode of the MOSFET conducts and pulls the source of the MOSFET down until VGS passes the threshold and the MOSFET starts to conduct. The source of the MOSFET is then further pulled down to a LOW level by this action of the Arduino taking the 5 V logic level LOW. So both the Arduino and the GPS module go LOW to the same voltage level.

The three states explain above show that the logic levels are transferred in both directions, independent of which device is driving the logic level.

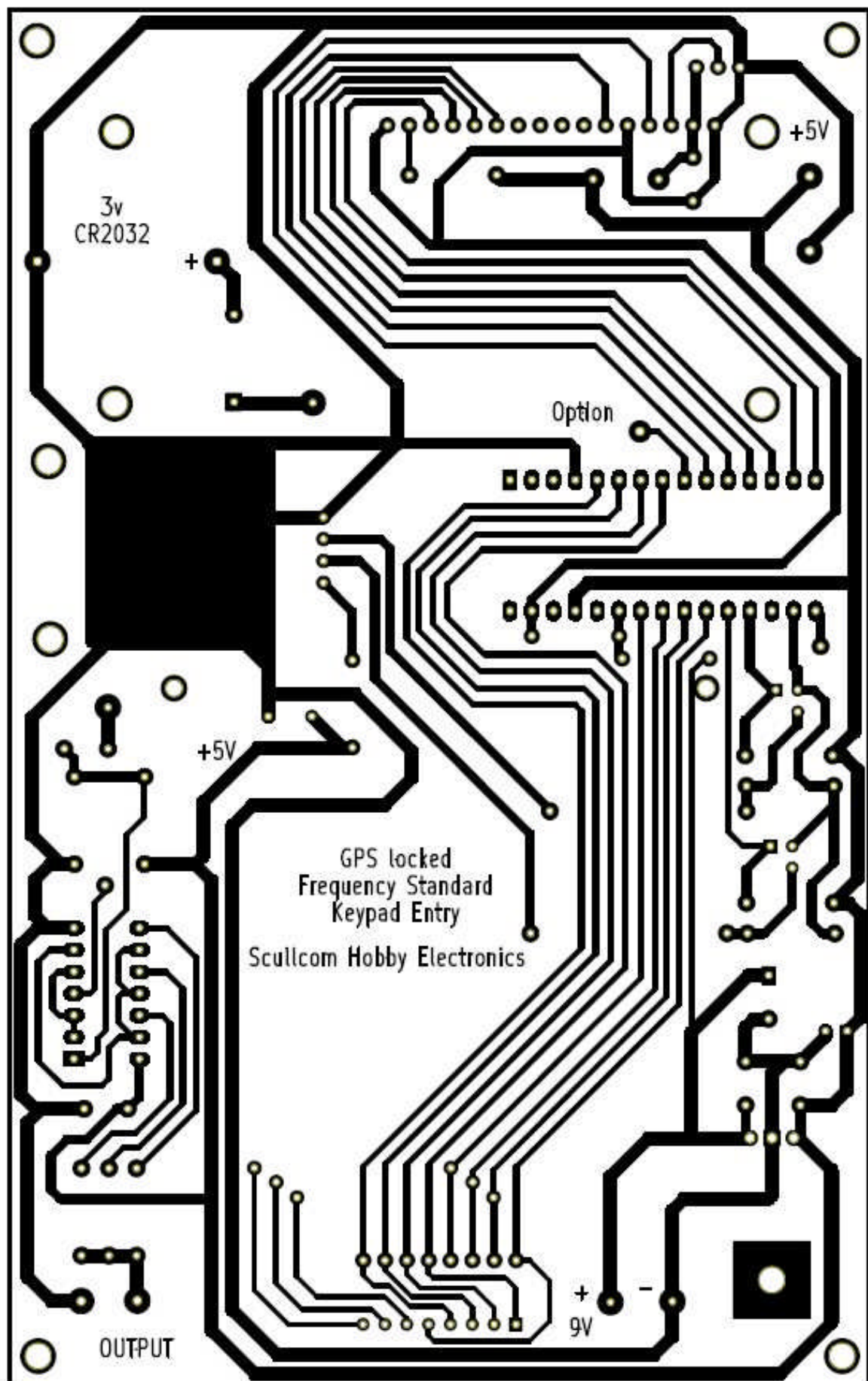
[illegible]



Component layout



Component layout with LCD and Keypad installed



Printed circuit board layout

Reading the GPS module data to extract the number of Satellites detected

When the GPS module starts receiving satellite data it provides a constant stream of data sentences which are known as NMEA sentences. Within this data stream the most important NMEA sentences include the GGA which provides the current Fix data, the RMC which provides the minimum GPS sentences information, and the GSA which provides the Satellite status data. In our project we will use the GGA data stream to extract the number of satellites detected and locked on to.

Below is an example of a typical GGA data sentence:

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

Where:

GGA	Global Positioning System Fix Data
123519.00	Fix taken at 12:35:19 UTC (current time)
4807.03830,N	Latitude 48 deg 07.038' N
01131.00000,E	Longitude 11 deg 31.000' E
1	Fix quality: 0 = invalid, 1 = GPS fix (SPS), 2 = DGPS fix, 3 = PPS fix, 4 = Real Time Kinematic, 5 = Float RTK, 6 = estimated (dead reckoning) (2.3 feature), 7 = Manual input mode, 8 = Simulation mode

08 Number of satellites being tracked

0.9	Horizontal dilution of position
545.4,M	Altitude, Meters, above mean sea level
46.9,M	Height of geoid (mean sea level) above WGS84 ellipsoid
(empty field)	time in seconds since last DGPS update
(empty field)	DGPS station ID number
*47	the checksum data, always begins with *

We will be using the data highlighted in **purple** above to show the number of satellite on our LCD. In order to extract this information we will set up a second Software Serial port.

First we need to identify the NMEA data we need to capture and we do this with the flowing 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
```

When the characters "\$GPGGA," are detected it starts reading the remainder of the string data in to the buffer. So the captured data starts from the first digit of the 'time' data (and not from the \$ character). The captured data then looks as follows:

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

With in the main loop we will use the substring command to extract the number of satellites from the GGA data sentence. This is a simply and easy way of extracting the data you wish to use. The substring command is defined as follows:

`substring(index, to)` – this returns a string with the characters starting from **index** and ending at the character location before the **to**

So to extract the number of satellites (highlighted in read) from the GGA data sentence below

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

Position 0 Position 39

We use the following code:

```
sats = GPGGAdata.substring(39,41);
```

The string "sats" now is equal to "08"

If you wanted to extract other data you simply need to change the "index, to" numbers in the substring. For example to extract the current time to a string labeled time then you would use:

```
time = GPGGAdata.substring(7,13);    //the string "time" now is equal to "123519"
```

Details of Code used in program loop to extract number of satellites and display on LCD

//new variables added at start of program

```
String sats;                //number of satellites detected
String GPGGAdata;           //satellite data string from GPS module
String FreqStatus;          //used to print Frequency locked if applicable
int satNum;                 //number of satellites detected as an integer
```

//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 nonexistent 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 integer 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 GPGGGA data sentence printed and then carriage return
    Serial.print("Satellite = ");
    Serial.print(satNum);
    Serial.println();
```

```

Serial.print(FreqStatus);
Serial.println();

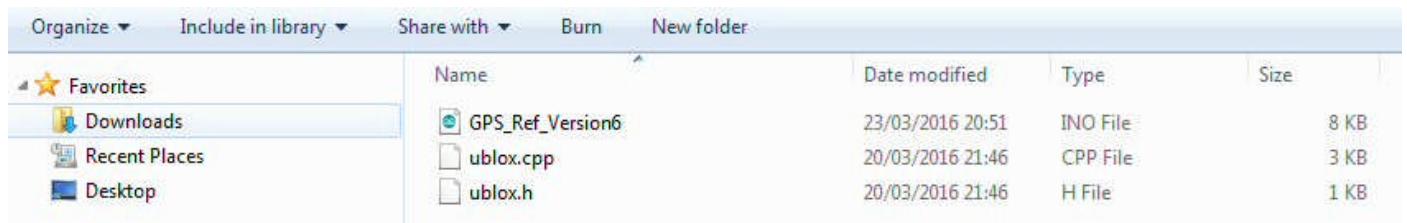
lcd.setCursor(13,0);           //set cursor position to column 13 of row 0 (first line on LCD)
lcd.print(" ");               //clears number if no satellites detected (3 spaces used)
lcd.setCursor(13,0);           //set cursor position to column 13 of row 0 (first line on LCD)
lcd.print(satNum);             //print number of satellites to LCD
lcd.setCursor(0,1);            //set cursor 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

```

The software for this project is made up of three parts all of which are include in a download zip file.



Once the file is unzipped you will have a folder as above. In this folder you will find the following files shown below:

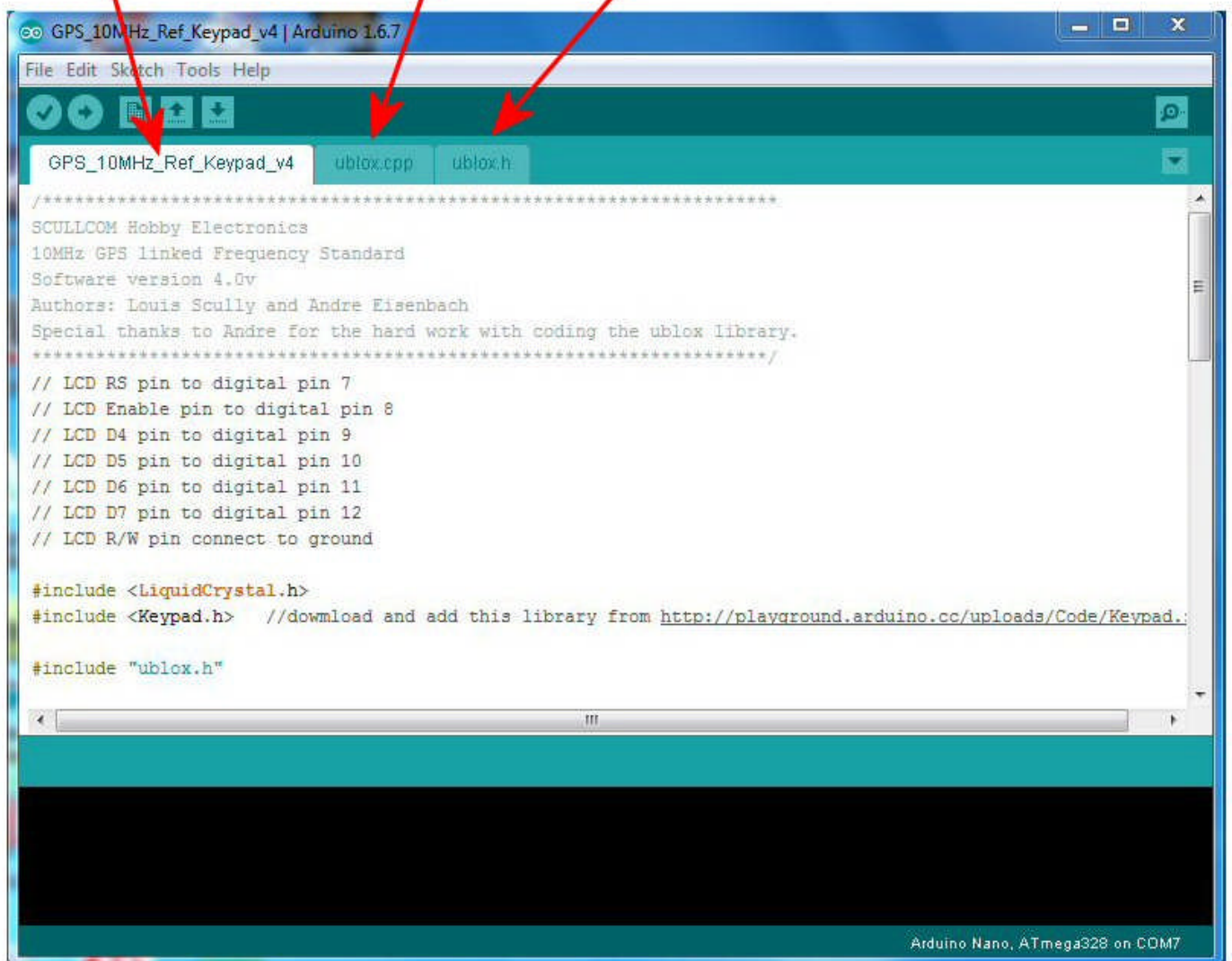


Please ensure that you are running the latest version of the Arduino IDE which is “Arduino 1.6.7” (older versions may not work). Run the “GPS_Ref_Version6” file and you should find the software opens up in the Arduino IDE as shown below:

Main Program

ublox.cpp

ublox.h



The screenshot shows the Arduino IDE interface with the title bar "GPS_10MHz_Ref_Keypad_v4 | Arduino 1.6.7". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". The toolbar contains icons for opening, saving, and running. The file explorer shows three files: "GPS_10MHz_Ref_Keypad_v4", "ublox.cpp", and "ublox.h". The main editor window displays the code for "GPS_10MHz_Ref_Keypad_v4". The code includes comments about the project, pin configurations for an LCD, and includes for the LiquidCrystal, Keypad, and ublox libraries. The status bar at the bottom indicates "Arduino Nano, ATmega328 on COM7".

```
GPS_10MHz_Ref_Keypad_v4 | Arduino 1.6.7
File Edit Sketch Tools Help

GPS_10MHz_Ref_Keypad_v4 ublox.cpp ublox.h

/*****
SCULLCOM Hobby Electronics
10MHz GPS linked Frequency Standard
Software version 4.0v
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>
#include <Keypad.h> //download and add this library from http://playground.arduino.cc/uploads/Code/Keypad.
#include "ublox.h"

Arduino Nano, ATmega328 on COM7
```

Now compile and upload the software to the Arduino nano on the main PCB.

/******

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> //download 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 {
  GPSTransceiver.flush();
  if (GPSTransceiver.available())
  {
    GPSTransceiver.read();
  }
  char gpsin[8] = "$GPGGA,";
  if (GPSTransceiver.find(gpsin)) {
    GPGGAdata = GPSTransceiver.readStringUntil('\n');
    sats = GPGGAdata.substring(39,41); //extract number of satellites data
  }

  satNum = sats.toInt();          //convert sats string to an integer 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 cursor 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 cursor 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 cursor 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> // for memset()
```

```
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

```
