Saturday, June 28, 2014

Advance: u-blox Neo-6M GPS Module

(UPDATE: I made another tutorial on this module which addresses the problem that you might encounter after trying out the method below. It still works, but not always. Read more if you are interested.)

Hi! Hopefully all of you are in high spirit because today we are going to explore a very exciting component, which is a GPS module! Well, I know this is actually an advance tutorial and probably too sudden for the second post of this blog, but this is what I'm currently working on. So, it is fresh on my mind and I'm very excited to share it with all of you.

This is a product that I bought from GI Electronic. You may view the product page here.

Introduction

GPS stands for Global Positioning System. It's in your phone, tablet, car and almost everywhere nowadays. Learning how to use this may also help you to find some ideas for your Final Year Project. It can help us to determine where the module's exact location in terms of latitude and longitude. We can also get some other information like the altitude, speed, time and so on.

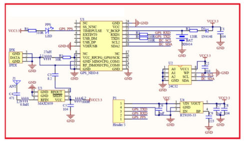
The module that we are going to program is using a very reliable and powerful chip, the u-blox Neo-6M. I love how this module works because it is very easy to hook up to the Arduino as it is using UART interface.
There's also no need for an antenna as it is already built-in on the module, unless you are going to use it indoor where there's many floors above you or you want to increase the gain. In that case, you'll need to attach an external active antenna to aid the connectivity.

What you need

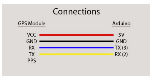
1. u-blox Neo-6M GPS Module
2. Arduino Board (I'm using Arduino Nano for this one)
3. Jumper wires
4. Breadboard
5.

Connections

This module is operating at 3.3V but if you refer to the datasheet provided (Below), you may notice that there's already a voltage regulator to convert to 3.3V and there's also a resistor connected to both RX and TX pins, so there's no worry in connecting this module directly to the Arduino.

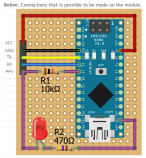


The required connections:



Above: Required connection. PPS pin is optional. RX and TX pins on the Arduino may change according to your preferences.

Below: Connections that is possible to be made on the module.



What is the purpose of PPS pin?

The PPS pin can be connected to a digital pin to detect when the GPS module obtain a fix. Just make sure to put a 10kΩ resistor as well in between the PPS pin and the digital pin. I don't know if it is really necessary, but one more step to protect your module and Arduino board won't hurt, right. The PPS pin will go HIGH for 900±1 milliseconds and LOW for 100±1 milliseconds when it gets a fix and will go LOW for as long as the module didn't get a fix. So, there'll be some extra condition for you to check. But, the good thing is, you'll be able to tell your program when the module get a fix. (However, in GPGGA NMEA message, there's already a Position Fix Status Indicator to tell you the position fix status.)
You may also connect this pin to an external LED to easily see the status. Let say your project's circuit board is quite complicated and obstructing the built-in LED on the module, this feature will be helpful. Just connect an LED and a suitable resistor to 3.3V and PPS pin like in the picture above.

The following code can be use to test out the input from the PPS pin according to the picture above.

```
#include <UBlox.h>
int input = 4;
bool last;
long lastTime;

void setup() {
  Serial.begin(19200);
  pinMode(input, INPUT);
  lastTime = millis();
}

void loop() {
  bool current = digitalRead(input);
  long currentTime = millis();

  if(current != last) {
    Serial.print(current ? "HIGH" : "LOW");
    Serial.print(F(" for "));
    Serial.print(currentTime - lastTime);
    Serial.println(F(" msec"));

    last = current;
    lastTime = currentTime;
  }
}
```

Code [Setting up the module and getting the raw data using library]

You should take a look at the tutorial that is attached on the product page first. The following code is derived from the tutorial. What I did was just compiling it all together into a library.
In the following code, I'm changing the setting from the default baud rate, 38400 to 9600 as it is too fast for the Arduino Nano to catch up. I'm also using pin 2 and 3 as RX and TX respectively. The module runs at 4Hz data rate in pedestrian mode. (More on this later)
***NOTE:** This library is using SoftwareSerial, so you need to import it in the main sketch. The limitation of SoftwareSerial also applies here.

Download "UBlox.h" library here.

```
#include <UBlox.h>
#include <SoftwareSerial.h>

UBlox gpsModule(2, 3, 9600);

void setup() {
  Serial.begin(19200);
  Serial.println("Initializing...");
  gpsModule.init();
}

void loop() {
  if (gpsModule.available()) {
    Serial.print(gpsModule.read());
  }
}
```

About the library

Constructor:

```
UBlox(byte rx, byte tx, unsigned int baudrate, [byte mode], [byte rate]);
```

rx = RX pin
tx = TX pin
baudrate = GPS baudrate
mode = optional, see Navigation Mode below
rate = optional, see Data Rate below

NAVIGATION MODE

Value	Description
0	Portable
1	Stationary
2 (Default)	Pedestrian
3	Automotive
4	Sea
5	Airborne < 1G
6	Airborne < 2G
7	Airborne < 4G

Search This Blog

Search

About Me

Ahmad Shahril
Follow 71
I'm a simple guy...Simple enough not to be known by anyone...
View my complete profile

Popular Posts

- Advance: u-blox Neo-6M GPS Module
(UPDATE: I made another tutorial on this module which addresses the problem that you might encounter after trying out the method below. It...
- Intermediate: QC12864B (128x64 LCD using u8glib.h)
Hi, everyone. Today, I have some time to kill. So, I decided to play around with my 128x64 pixels LCD. This is also a product distributed b...
- Intermediate: u-blox Neo-6M GPS Module (Revisited)
Previously, I've posted about this module which you can find here . This time around I'm going to share with you another library th...
- Basic: HMC5883L (Digital Compass)
Hi, everyone. I am actually waiting for the time for breakfast. I'm writing this post to take my mind off the meal. So, this time, it&#...
- Basic: HC-06 Bluetooth Module
Hi. It has been a while since my last post. I was busy with the Eid preparation, getting my driving license, babysitting, etc. Finally I h...
- Intermediate: Getting Started with ATtiny13A-PU
Hi, everyone. It's been sooooo long since I last posted a new blog. Mostly because I'm busy and lazy. :D Anyway, I'm in a g...
- Upcoming Post: LED Cube
Hi! It has been too long since I last wrote a post on this blog. I'm currently working on multiple projects and really busy. This is ...
- Basic: Getting Started [Arduino]
Well, actually, I don't have time for this kind of post. There's a lot of information in getting started with Arduino already...

Blog Archive

► 2015 (2)
▼ 2014 (6)
► September (2)
► July (2)
▼ June (2)
Advance: u-blox Neo-6M GPS Module
Basic: Getting Started [Arduino]

Labels

Arduino (7) Intermediate (4) Basic (3) GPS (2) SPI (2) uBlox (2) 74HC595N (1) ATtiny13 (1) Advance (1) Bluetooth (1) HC-06 (1) HMC5883L (1) LCD (1) System Integration (1) TinyGPS (1) digital compass (1) u8glib (1)

DATA RATE

Value	Description
0	1Hz
1	2Hz
2	3.33Hz
3 (Default)	4Hz

Functions:

init() : To initialize the GPS module with the settings provided.
getBaudrate() : Return the GPS baud rate.
available() : Returns 'true' if the GPS has new data. Else, returns 'false'.
read() : Returns a character from the serial communication.
*sendUBX_Msg(byte *UBXmsg, byte length)* : To set more UBX command. Return 'true' if command successfully sent and 'false' otherwise.
UBXmsg is the UBX command string in form of array of byte.
length is the *sizeof(UBXmsg)*.

Example to use *sendUBX_Msg*:

```
byte msg[] = {0xB5, 0x62, 0x06, 0x16, 0x08, 0x03, 0x07, 0x03, 0x00, 0x51, 0x08, 0x00, 0x00, 0x8A};
if(gpsModule.sendUBX_Msg(msg, sizeof(msg))) {
  Serial.println("Successfully set SBAS setting...");
}
else {
  Serial.println("Failed to set SBAS setting...");
}
```

Code [Process data using TinyGPS]

TinyGPS is well known for making it easy to process GPS data. You can download it from the following website: **http://arduiniana.org/libraries/tinygps/**

The following code is the integration of the UBlox.h library and TinyGPS.h library:

```
#include <UBlox.h>
#include <SoftwareSerial.h>
#include <TinyGPS.h>

UBlox gpsModule(2, 3, 9600);
TinyGPS gps;

void setup() {
  Serial.begin(19200);
  Serial.println("Initializing...");
  delay(1000);
  gpsModule.init();

  byte msg[] = {0xB5, 0x62, 0x06, 0x16, 0x08, 0x03, 0x07, 0x03, 0x00, 0x51, 0x08, 0x00, 0x00, 0x8A};
  if(gpsModule.sendUBX_Msg(msg, sizeof(msg))) {
    Serial.println("Successfully set SBAS setting...");
  }
  else {
    Serial.println("Failed to set SBAS setting...");
  }

  byte msg2[] = {0xB5, 0x62, 0x06, 0x09, 0x0D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x21, 0xAF};
  if(gpsModule.sendUBX_Msg(msg2, sizeof(msg2))) {
    Serial.println("Successfully saved all setting...");
  }
  else {
    Serial.println("Failed to saved all setting...");
  }
}

void loop() {
  bool newdata = false;
  unsigned long start = millis();

  // Every 5 seconds we print an update
  while (millis() - start < 5000) {
    if (gpsModule.available()) {
      char c = gpsModule.read(); // comment to see raw GPS data
      if (gps.encode(c)) {
        newdata = true;
        // break;  // uncomment to print new data immediately!
      }
    }
  }

  if (newdata) {
    float latitude, longitude;
    unsigned long age;

    gps.f_get_position(&latitude, &longitude, &age);
    Serial.print("Latitude: ");
    Serial.println(latitude, 6);
    Serial.print("Longitude: ");
    Serial.println(longitude, 6);
    Serial.print("Age: ");
    Serial.println(age);
    Serial.print("Altitude: ");
    Serial.println(gps.f_altitude(), 6);
  }
}
```

Changing the Settings

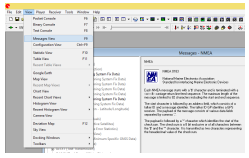
You should change the settings according to your needs. In the example above, I'm using Pedestrian mode, 4Hz data rate at 9600 baud rate. It is probably a good time now to open the **protocol file attached** on the product's page. Yes, it is a 222 pages document.

But, do not worry. For me, either I don't understand or I just don't read 85% of the document. You just need to understand a few things, unless you really want to make the best out of your GPS module, go for it! Read ALL of them. For me, it is good enough to know a few things. Below are a few things that I think you should read about in the protocol documentation.

1. You should know all of the different types of navigation modes provided to set the most suitable setting for your project. They are listed in the *Navigation Mode* table above and you can also find the descriptions on page ?.
2. You should also read about some of the UBX protocols. UBX protocols are very important as they are the one responsible in setting up most of the settings for the GPS module. I recommend you to read about UBX structure, ACK, CFG-MSG, CFG-NAV5, CFG-NMEA, CFG-PRT, CFG-RATE, and CFG-SBAS.
3. When dealing with GPS, you should also know about NMEA messages, and the most important one you should know is about GPGGA. (Page 56).
Now that you've read about those things, let's try to change some of the settings.

If you pay attention and read about the UBlox library, you may notice of the example for using the function *sendUBX_Msg(byte *UBXmsg, byte length)*. However, how would you know the command string that you should supply to the function? Well, the answer is, use the software provided by u-blox, which called **u-center**.

The software is quite powerful. You can analyse your GPS data in one place. You could even set the GPS settings directly. After you've installed the software, launch it and press F9 or go to *View > Messages view*.



Let say we want to change the SBAS settings. Expand *UBX > CFG > SBAS (SBAS Settings)*
Change the settings using the interface on the right. When you're finished making your changes, copy the HEX code below it. Make sure that you only copy the ones in range as shown below:

Paste it in the *setup* function of the Arduino. Make some changes so that it turns from:

```
B5 62 06 16 08 00 01 07 01 00 00 00 00 00 20 C9
```

into array of byte like the following:

```
byte msg[] = {0xB5, 0x62, 0x06, 0x16, 0x08, 0x03, 0x07, 0x03, 0x00, 0x51, 0x08, 0x00, 0x00, 0x8A};
```

You can change the array name as you like. Next, we can call the function *sendUBX_Msg(byte *UBXmsg, byte length)*. I'm assuming that we've already created a UBlox instance called *gpsModule*:

```
byte msg[] = {0xB5, 0x62, 0x06, 0x16, 0x08, 0x03, 0x07, 0x03, 0x00, 0x51, 0x08, 0x00, 0x00, 0x8A};
gpsModule.sendUBX_Msg(msg, sizeof(msg));
```

What if you wanted to change the data rate or the navigation mode? Simple. Just add two more parameters in the constructor like below. Refer to the tables above for the code.

```
UBlox gpsModule(2, 3, 9600, 1, 3);
```

The code above will results in stationary navigation mode and 4Hz data rate.

Conclusion

The GPS module that we used in this tutorial is very easy to be used now, with the existence of the libraries. However, the UBlox.h library is still under Beta development stage. There might still be some bugs somewhere. If you find it, please let me know. Also, if you have any question you want to ask, please comment below.
I'm still learning and if you find something that is worth sharing, please do so. I may not be able to solve all of them, but someone might shed some light. I think that is all for now, have a nice day.

Labels: Advance , Arduino , GPS , TinyGPS , Tutorial , uBlox