A new Project: GPS + AVR ATMega328P + OLED = APRS Board !

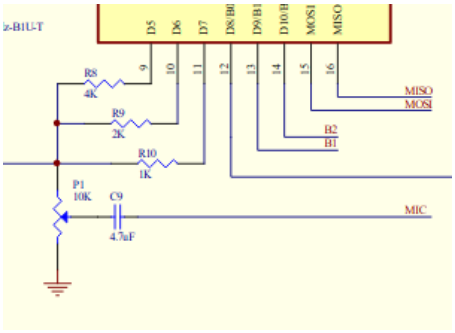
Hello to all, this is SV1ONW from Athens.

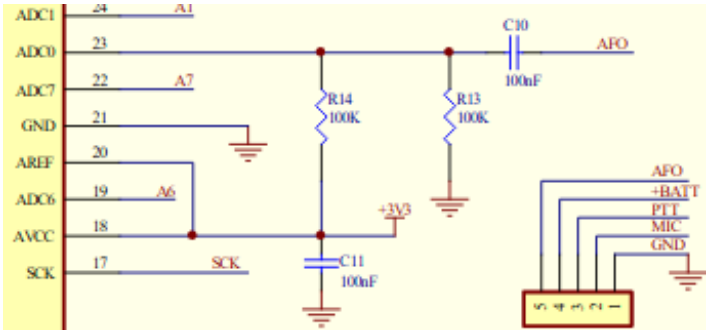
I was browsing SV1AFN's New Site (www.sv1afn.com), when my attention was caught by Makis' new APRS Controller Board that makes a nice companion to his DRA_818 V and U boards. Having already used one of his DRA_818/V for an experimental APRS tracker project of mine, I started looking at the board's schematic after reading the initial specs:

- Similar size to our DRA-818V and DRA-818U VHF and UHF transceivers
- Similar interface so they both can be integrated together to make a APRS VHF or UHF station
- Low-power design
- On board Li-Po Battery controller, charger with I2C battery capacity Fuel-Gauge meter
- With battery disconnect MOSFET
- Arduino compatible controller
- I2C interface for an external OLED display

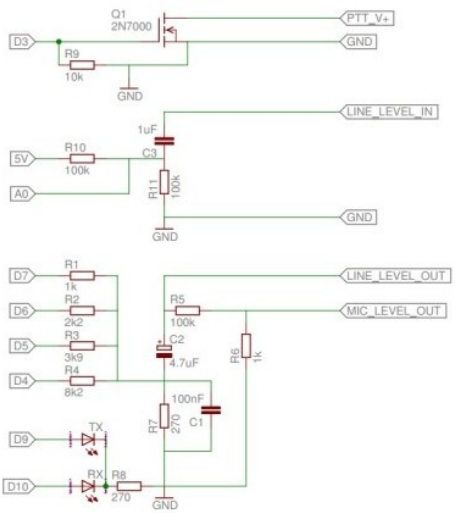
Examining the schematic (available at product's location) revealed an onboard ATMega328P-AU MCU (TQFP footprint) with an ISP 6-pin header, for programming the chip obviously. An extrenal 8 MHz crystal provides the timing to the MCU which operates on 3.3 Volts, offering direct interface to any i2c display without the intervention of level translators! That is great. Memory wise 2KB RAM / 32KB flash of the ATMega328P-AU looks promising for quite a number of projects.

The circuitry around the MCU revealed that there was an APRS type of Micromodem,

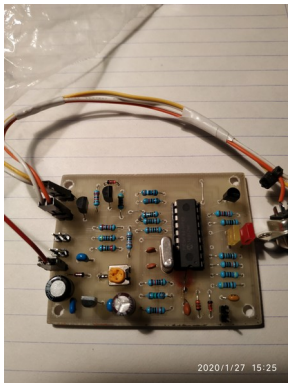




based on the standard of Mark Qvist's (http://unsigned.io) as shown on the following schematic:

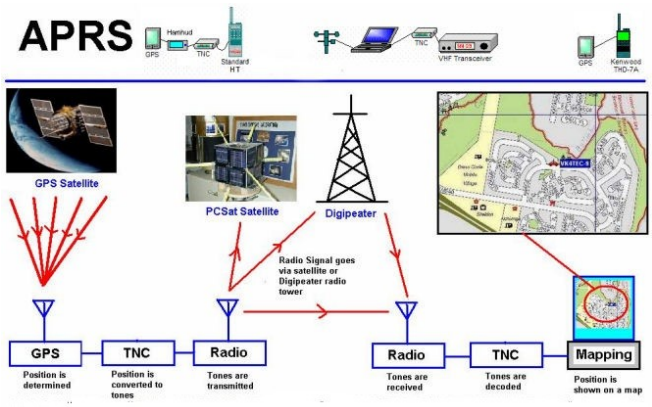


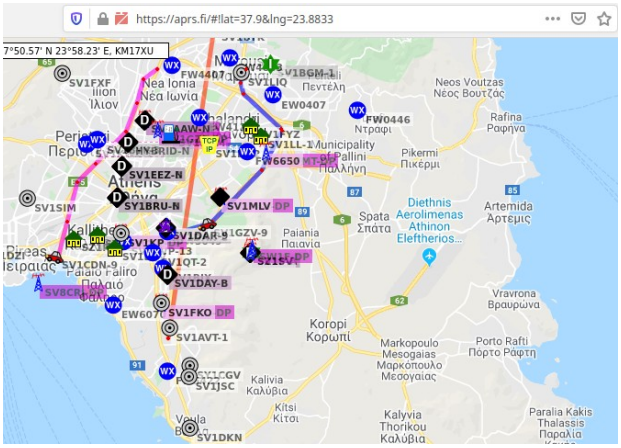
Long ago I had built a TNC Modem based on a PIC 16F88, and then just a bare board with the same MicroModem circuitry, to be used with an Arduino board. Quite a popular concept.



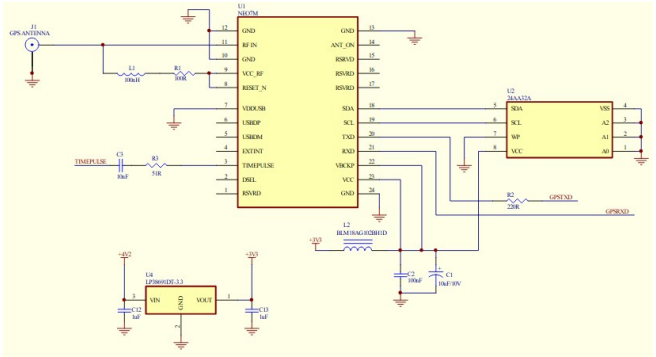
MicroModem (Wikipedia definition) is an open-source implementation of a 1200-baud AFSK / Bell 202 modem on the popular ATmega328p microprocessor. MicroModem can be used for things like ham radio APRS, AX.25, TCP/IP over SLIP, experimentation with mesh-networks, long-range wireless communication with sensors (Or friends! Or strangers!). LORA of course is another possibility. The modem basically comes in two flavors: Prebuilt or DIY.
I am always thinking DIY-wise. But before taking it to the next stage, a bit more of Ham Radio history. Remember the bulky old TNC modems? Like the MFJ-1278B. Big box!
A **terminal node controller** (**TNC**) is a device used by amateur radio operators to participate in AX.25 packet radio networks. It is similar in function to the Packet Assembler/Disassemblers used on X.25 networks, with the addition of a modem to convert baseband digital signals to audio tones. The TNC was originally developed by Doug Lockhart, VE7APU, of Vancouver, and popularized by the Tucson Amateur Packet Radio association with the TNC-1 and TNC-2. With modern technology just the above circuitry with an Arduino class MCU is enough to play the trick and build a portable APRS tracker.

What about APRS? No, no, this is not a tutorial for Automatic Packet Reporting System (aka. APRS). Please look it up. I've just added some teasing pics.

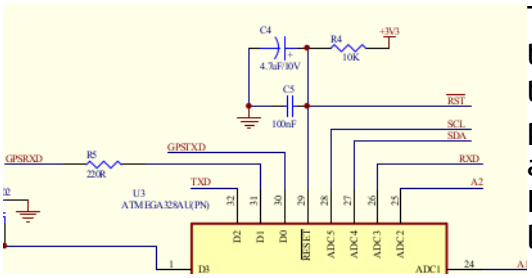




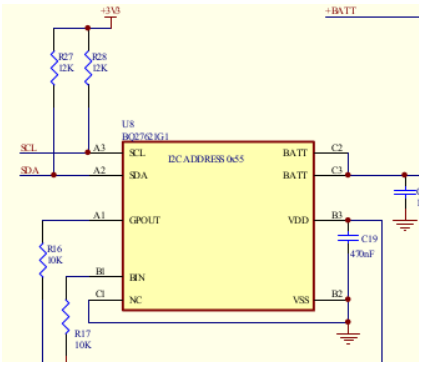
Going back to the APRS board now, the ATMega328P has interface to a very modern GPS chip with external memory that communicates through the standard AVR/Arduino UART to the MCU to provide Global Positioning data through the GPS Sattelite system, such as Location Lattitude and Longtitude, Speed, Altitude and very accurate timing (Date & UTC Time) among others.



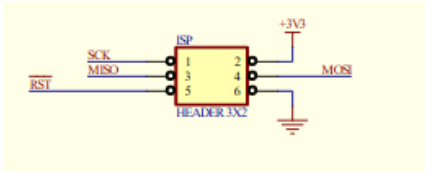
The GPS chip module is a Blox NEO-7M-0-000 that has a plethora of features for all Global Positioning Sattelite systems. As it connects to the UART pins (D0, D1) it is initiated through the Arduino function: Serial.begin() and the Baud rate recognized by the NEO-7 module is 9600 Bps.



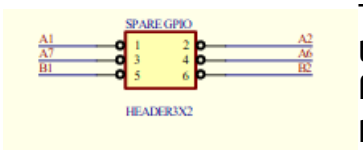
The next thing that I noticed was the interface with the Dorji DRA818/V or /U module. As you can see in the schematic, communication is carried through pins D2 (TXD) and A3 (RXD), but this is no problem as MCU's analog pins can be used as digital ones. For such a communication Arduino's SoftSerial library should be used.
To test the DRA818/V module with the APRS board, I prepared a small sketch file which can be uploaded to the MCU. It simply programs the TX and RX frequencies of the tranceiver module to 144.800 MHz which is the IARU Region 1 APRS frequency for most countries of that region. Please bare in mind that in order to use/transmit through an APRS transceiver **you must have a valid Radio Amateur's liscence and a callsign**. The test sketch invokes the PTT circuit through a push button for Tx.
Details about that are included as comments in the code.
Last but not the least, on the above schematic you can see the two analog lines A4 (SDA) and A5 (SCL). These are the i2c lines that can be used for connecting an OLED or LCD display. Moreover on these lines the Battery Gauge chip is connected.



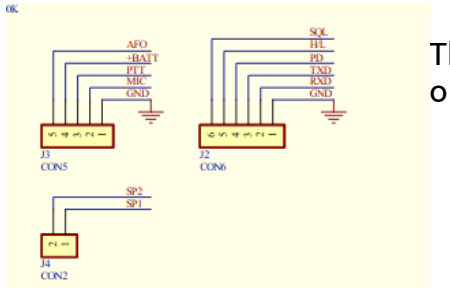
What is left on the MCU area of the board are the headers (connectors) that are used for programming the board and the spare pins that are available for future options.



This is the ISP 6-pin header (in-circuit serial programming) through the SPI interface.
As there is no actual USB interface on the board, like in an Arduino Uno or Nano, the loading of code / programming of the APRS board is carried out through this header, using a Programmer or an Arduino Uno or Nano board as programmer. One thing to clarify, the USB connector on the board exists only for powering the board in standalone mode or for charging the LIPO battery cell. This is no problem.

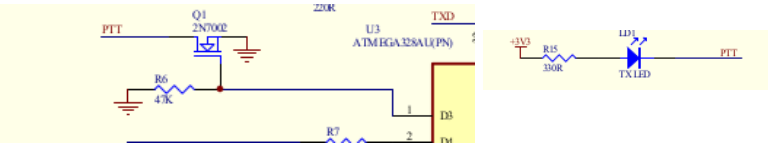


There is another 6-pin header next to the ISP one, which is for the remaining spare GPIO pins of the board, two digital and four analog. They are very handy, as they can be used for many pursposes. Two af them could be used to connect a USB FTDI to TTL RS-232 adapter, should there be ever need for one.

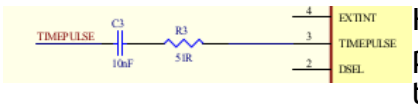


Then we have the headers for the piggy-back connection of the DRA818 VHF or UHF board on the APRS board. This

is a very convenient feature making the interconnection of the two boards a very easy task. Just snap them.

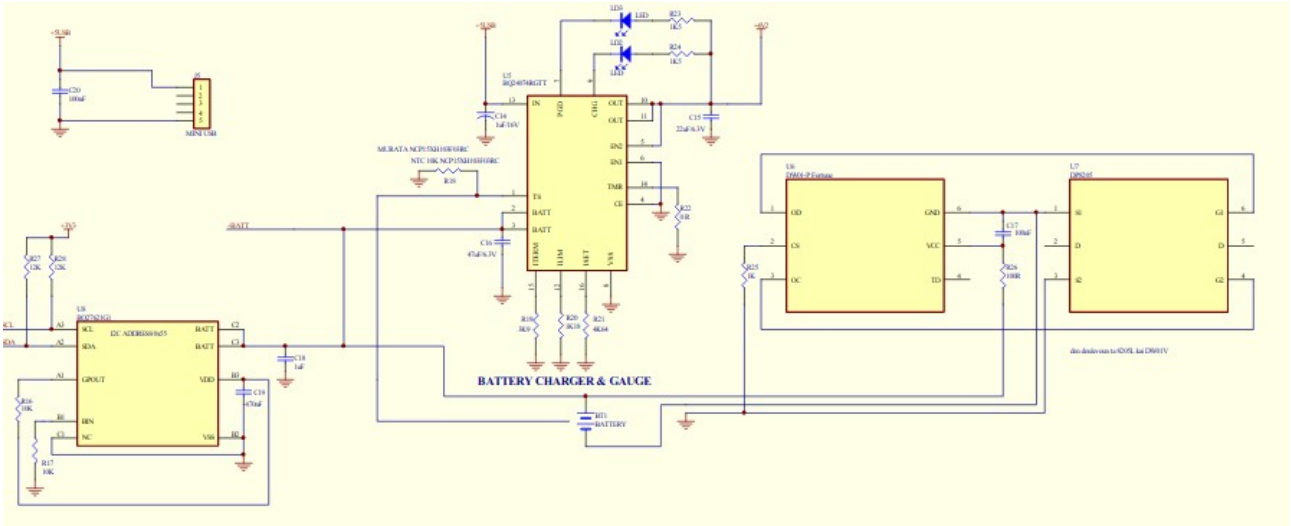


We also have 2N7002 MOSFET for controlling the PTT line of the transceiver board, as well as an on-board LED indicator which is switched on when the PTT line is active. On the left side of the board there is a 2-pin header for connecting a handy push button on digital pin D8, that can be programmed for many purposes. Next to it there is an empty pad labeled “TP” which by tracing the schematic I found it to be the 1pps output of the GPS module that can be also useful for syncronization purposes.



Having the i2c pins available and one of the spare digital pin, one could use them to connect, say an Si5351a module, to make a GPS discipline controled PLL fixed, or even variable generator. Remember the remaining spare pins, why not assigning two of them to a Rotary Encoder? Yes, it is possible. Just food for thought!

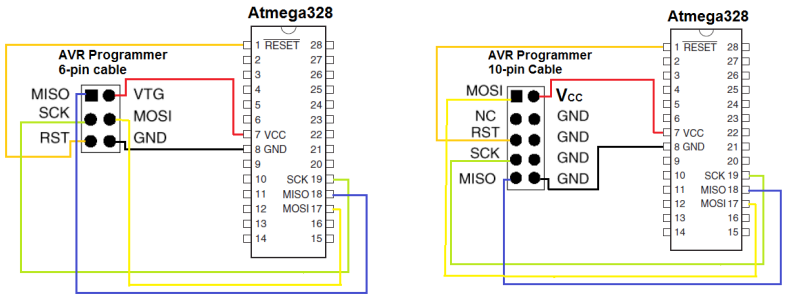
I explained the use of the A4 and A5 analog pins for connecting i2c peripheral devices (SDA, SCL signals). But how do you power those modules? These are just two signal pins. It is quite simple. There are two pins on the ISP header that provide +3.3 V and ground. The ISP cable is connected only for uploading the code to the MCU, then it is removed. So you can use these pins for powering i2c modules. There are explanatory pics below. There is a hefty on-board 3.3 V regulator and next to the i2c header there are two extra pins for connecting the LIPO battery.
SV1AFN has designed on the board a very smart circuit for powering the board and SAFELY charging the battery with two LED indicators. One of them is on when the board is powered from the micro USB connector and the other when the LIPO battery is attached to the board. Finally there is the Texas Instruments BQ27621G1 integrated circuit Gauge meter (U5 on the schematic below) that provides true battery State of Charge (SOC) to its i2c pins that connect to the MCU, plus more functions.



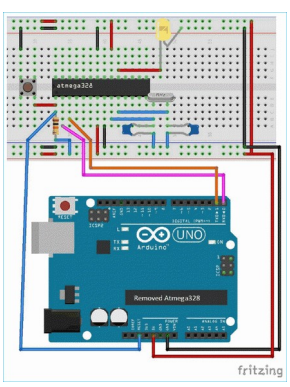
This is not just a plain charging circuit, it is a full “Battery Factory”, where even the chemistry of the LIPO battery being used can be defined in the code.
Next to the GPS “stamp size” module there is a very tiny coaxial connector for the external GPS antenna. I am not sure of its type, or name.
I think I almost covered everything that the schematic revealed to me, so the next logical question is how do you program Makis’ board?

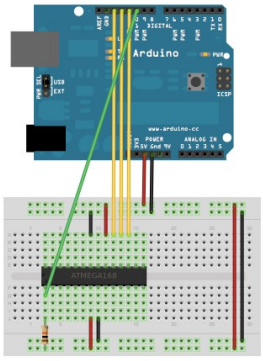
This is not a project board for a beginner! Or let me put it in another way. If you are a beginner and you want to play with something like that, do not try to fool yourself. I am not saying buy something ready-made, it is against my DIY for ever principle. But I would recommend to try this together with a friend that has some programming experience and could assist you and why not guide you to learn.
I assume that one should be familiar with at least the Arduino IDE (Integrated Development Enviroment) or equivalent tool and to have some experience in installing Libraries and compiling Arduino sketches. Code is written in C++ and the examples are documented as much as possible. If you want to duplicate the examples, there are chances that you may succed. But beyond that point you should know what you are doing or you should seek for help from someone that knows programming or has some experience with the Arduino ecosystem!

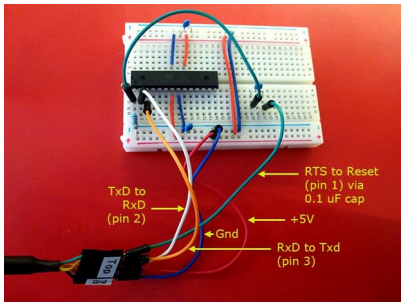
The APRS board comes blank from any code. So how do you upload (i.e. put) a program on it? Or even how do you add a bootloader to the blank ATMega328P MCU chip?
We will use the Arduino IDE, version 1.8.11 as of writing this article, or any other late previous version such as 1.8.9 or 1.8.10 which I used for compatibility test issues in my discussion.
The IDE communicates with your computer through a USB port connection, which is connected to the development enviroment (target) for uploading your code in the form of an ".ino" file, so called sketch. It could also be a hex file.
Many of the Arduino modules or boards such as the Uno, Nano, Mega do have a USB port for this purpose, but there are others, such as the MiniPro or the APRS board or a barebones ATMega328P/PU chip that do not have a USB port. In that case you need to use either a USB to serial converter such as the USB FT232RL FTDI board (TTL serial to USB, or use the ISP (sometimes called ICSP) header available on the board for programming purposes. A barebones ATMega328P/PU chip does not have any header, so you must add one yourself following one of the many available tutorials. It is quite simple, see the following drawings:



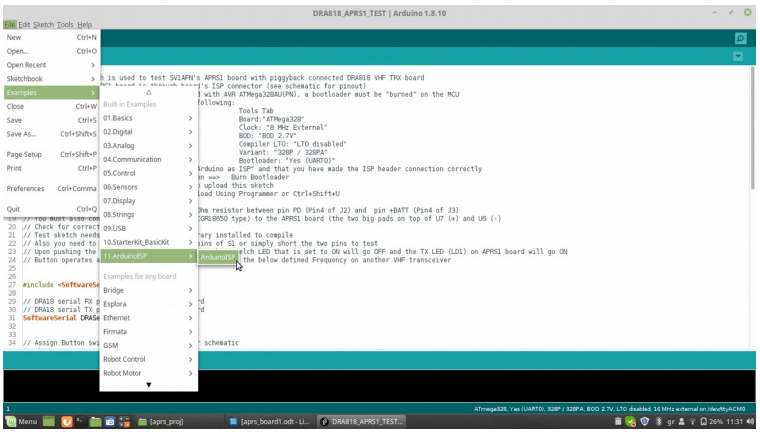
They cover most of the cases with the popular 6-pin and 10-pin header nomenclature. VTG or Vcc is the +3.3 V or +5.0 V positive power rail with reference to GND (Ground). Please notice that the first circuit in breadboard design is with no external clock crystal (using internal RC timing up to 8 MHz) and the second with exernal crystal of 8, 16 or 20 MHz. Remember that for using 16 or 20 MHz clock, you must power the MCU with 5 Volts! Both can be used, depending what you are aiming at.



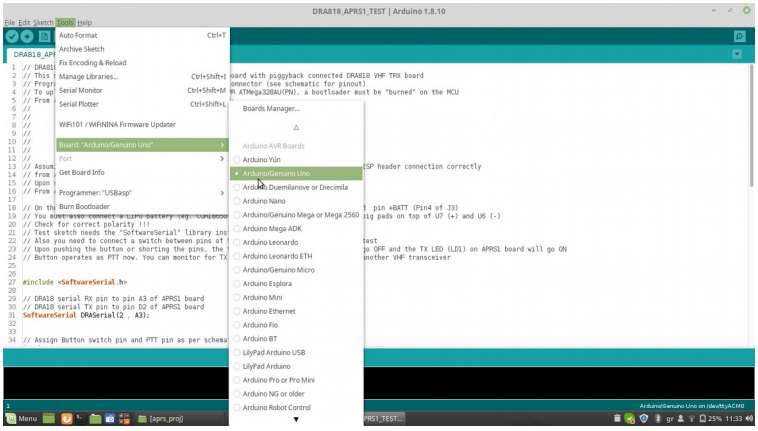




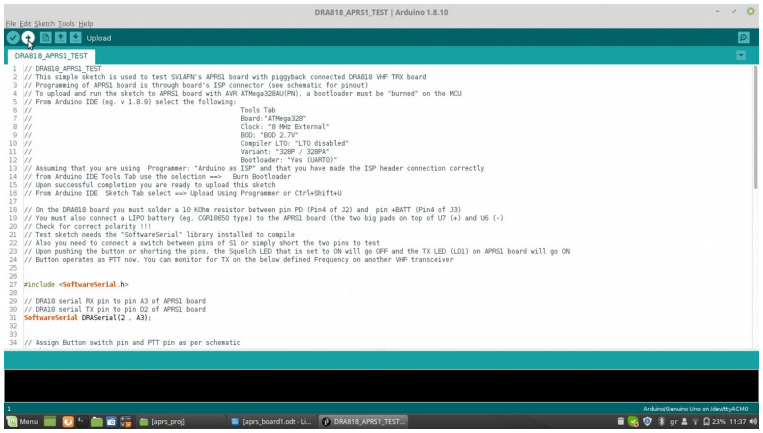
I hope that the above give you an idea for the barebones ATMega328P/PU case.
In the above examples we used an Arduino UNO that connects to your computer via a USB port, for programming the MCU, so we use the Arduino as Programmer. In order to do that sucessfully we must upload first to it the Programmer sketch from the examples available on the Arduino IDE.
That goes: **File>Examples>11.ArduinoISP>ArduinoISP** (Bold letters show the commands)



Then select the UNO board and the USB Port it is attached to (remember to connect the board). Select : **Tools>Board:>Arduino/Genuino_Uno**,

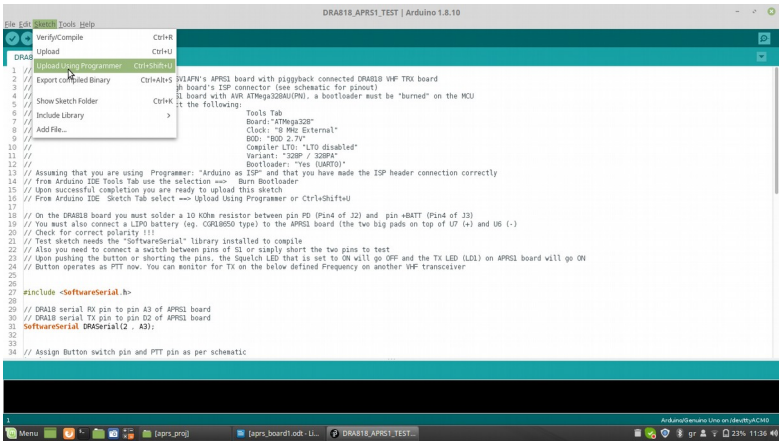


and then upload the selected sketch, by clicking the **upload button**.

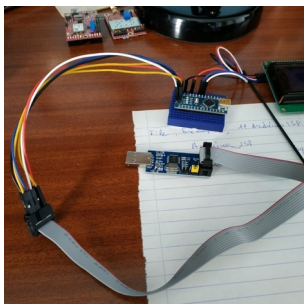


If you have followed the instruction correctly, you now have a working Arduino UNO as ISP programmer with blinking LEDs. You can then have it connected to the target ISP 6-

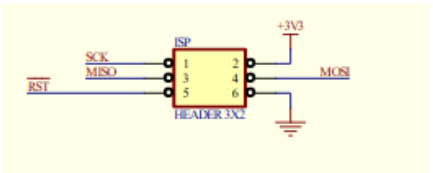
pin header and you are ready to upload the sketch of your choice, say the DRA818_APRS1_TEST.ino, using NOT the standard **upload** command or Button, BUT the command: **Upload using Programmer**.

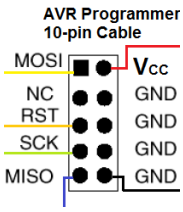


Alternatively of course you can use any programmer of your choice that you may have or you wish to build DIY style. For a change, I use the very inexpensive and small size AVR-ASP USB ATMega8 programmer that goes for a cost of aprox. 3 Euro (including 24% VAT). This one is really small and handy, plus that I can carry it around inside an Altoid tin box!

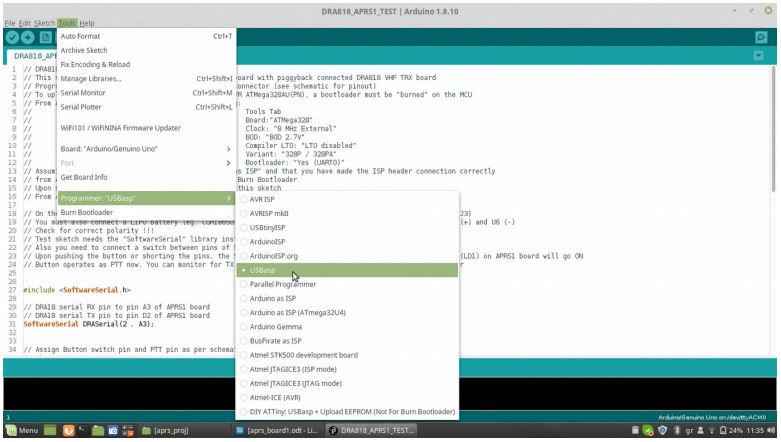


Looking at the pic you will agree that you cannot beat the cost and have one built DIY with less cost, period! Programming 10-pin header cable inclued.
If we are going to use this one, we do not need to download any sketch as it comes preprogrammed, ready to be used. It also has a jumper for selecting +3.3 V or +5.0 V for powering the target and you only need a 10-pin to 6-pin header converter that you can make yourself. In my pictured example I use it to program an Arduino Nano with damaged USB chip!
And here is the cheat sheet for connecting it to our APRS board 6-pin header.





That is all needed hardware-wise. I use 6 male to female single pin cables with different colors which I always store in the Altoids box together with the programmer! They are shown in the last picture. Notice that pin 2 (+3V3) and pin6 (GND) are the ones being used for powering the i2c device, a 0.96” 128 x 64 OLED for my first project. It is connected with a separate cable harness.
In the case of this programmer, all you have to do is select it in the IDE. It is the USB_ASP one.



Again in this case to upload the sketch of your choice, say again the DRA818_APRS1_TEST.ino, you must NOT use the standard **upload** command or button, BUT the command: **Upload using Programmer**. In other words, whenever you want to upload code through the ISP you must follow this procedure.
I see another issue. Upload to what device?
This is the next serious step. We have to follow the previous procedure of selecting a Board from the Tools tab and instead of Arduino/Genuino Uno selected previously for loading the programmer code, select just an ATMega328P MCU with an external clock of 8 MHz. If you look at the selection pull down menu you will not find such an option there. So we have to install it, following very closely the recipe described bellow.
There is code from Hans on his Github: https://github.com/MCUdude/MiniCore for that. It is called MiniCore and is an Arduino hardware package for definining ATmega8, ATmega48, ATmega88, ATmega168, ATmega328 and ATmega328PB MCUs in the IDE. If this is done, then you simply select the desired parameters for each MCU and you are in business!

Go to this Github and before doing, FIRST read carefully the “README.md” text. It is very nicely doccumented and explains the full story. I will focus only on the installation recipe which I have copied below.

How to install

Boards Manager Installation

This installation method requires Arduino IDE version 1.6.4 or greater.

- Open the Arduino IDE.
- Open the **File** > **Preferences** menu item.
- Enter the following URL in **Additional Boards Manager URLs**:

  `https://mcudude.github.io/MiniCore/package_MCUdude_MiniCore_index.json`

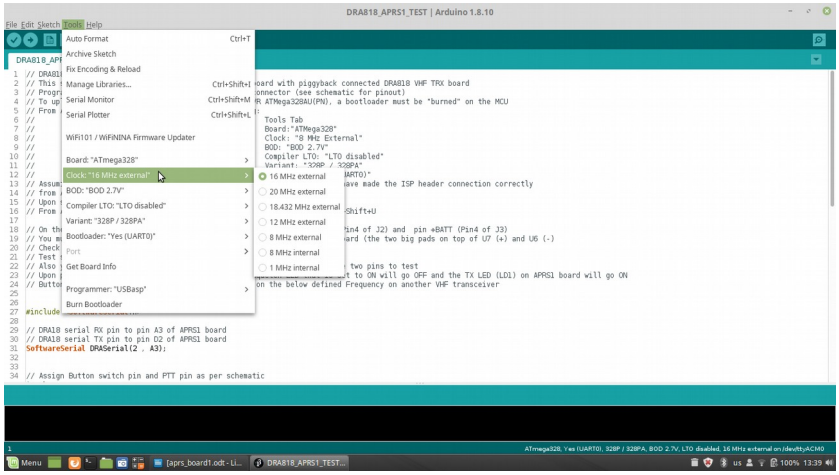
- Open the **Tools** > **Board** > **Boards Manager...** menu item.
- Wait for the platform indexes to finish downloading.
- Scroll down until you see the **MiniCore** entry and click on it.
- Click **Install**.
- After installation is complete close the **Boards Manager** window.

- **Note**: If you plan to use the *PB series, you need the latest version of the Arduino toolchain. This toolchain is available through IDE 1.8.6 or newer. Here's how you install/enable the toolchain:
    - Open the **Tools** > **Board** > **Boards Manager...** menu item.
    - Wait for the platform indexes to finish downloading.
    - The top is named **Arduino AVR boards**. Click on this item.
    - Make sure the latest version is installed and select it
    - Close the **Boards Manager** window.

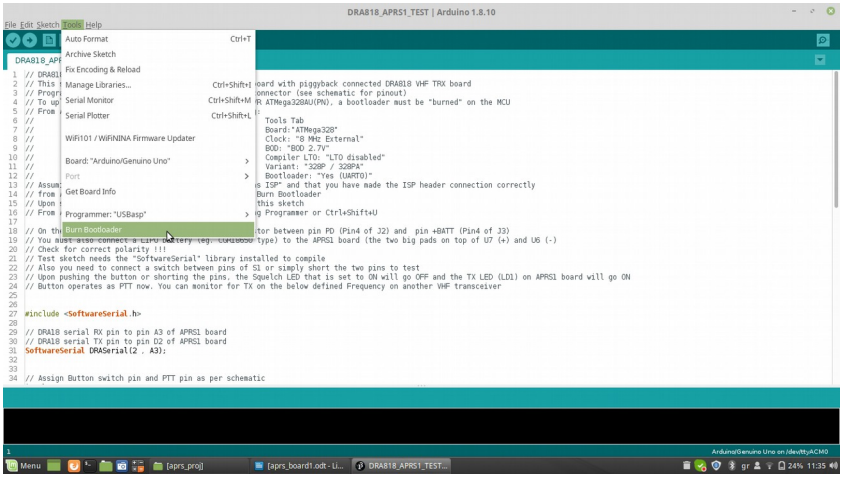
Now that we have it istalled, we can select the ATMega328P MCU with 8 MHz external crystal which is the one suitable for the APRS board from SV1AFN!

No matter which programmer we use, the next logical step would be to Burn a Bootloader to the MCU.

Now that we have defined our “Target board”, i.e. our MCU, it is very easy to go to the **Tools** tab and select the command: **Burn Bootloader**. Always make sure that the APRS board is correctly connected to the Programmer and the Programmer to your PCs USB!



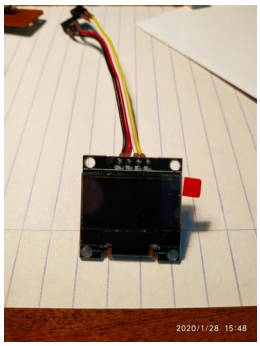
This is the dump of the MCU selection



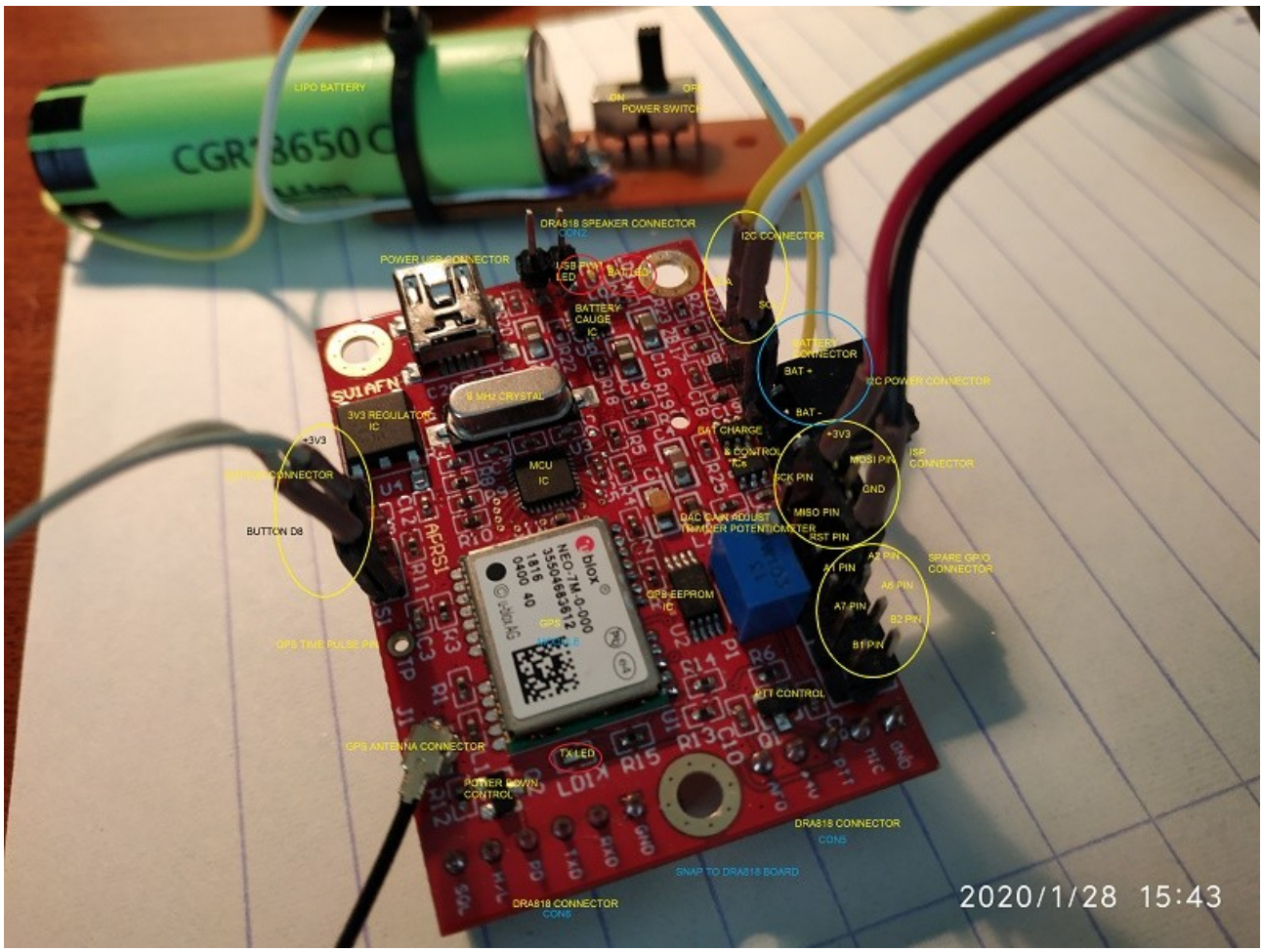
Here we are ready to Burn our Bootloader.
When we select the apropriate parameters for our MCU, if we decide or need to burn a Bootloader, the MCUdude/MiniCore knows which one to select. Bootloader is not really

necessary in the case of APRS board as there is no actual USB communication port on board.
Having tried to explain this whole programming process, we are ready to try upload using the programmer to the APRS board the DRA818_APRS1_TEST.ino, just to familiarize ourselves with the process and test the DRA818/V board, if we have got one. The sketch can be easily modified for the UHF module, by simply substituting the defined frequency with one in the 430 to 440 MHz band for IARU Region 1. Different frequency allocations apply for the other IARU Regions.
We can then test the reception by listening for the APRS AFSK coded signals on 144.800 MHz (bursts of sounds) and try to transmit by pressing the Push Button, assigned by the sktech for PTT Tx control. Use an appropriate antenna on the DRA818/V for both Rx and Tx. Do not expect to hear anything without an antenna, unless you transmit yourself with another Handheld transceiver in the same room.

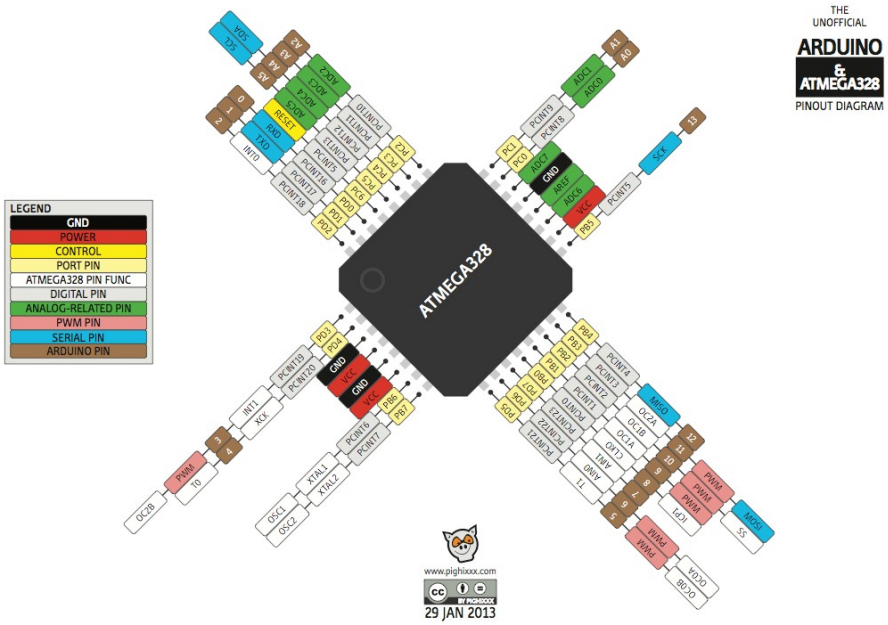
Now that we are ready, and things are working, having learnt how to Upload an ".ino" sketch on the APRS board let us try to build something more useful and meaningful.
I have many times read and heard about a Ham's Clock helper, so I thought that by just using the APRS board with its GPS module and an OLED display, I could simply build a battery powered portable device, good for the field day or the Ham shack. It would display your Maidenhead Locator and the UTC time, together with some more extra GPS data such as Longtitude, Latitude, Number of SATs available at the GPS receiver location and possibly Speed and Altitude, depending on your personal tastes and needs. The GPS NMEA sentence has a lot of data to retreive from and its quite easy to do so learning from the existing functions in the present code.
The program should be as compact as possible and leave space for future expansions such as APRS Tracking.
The hardware is nothing. Just plug the OLED, LIPO battery and GPS antenna and there you go!
Nowadays, I always use a 0.96inch 128 x 64 pixel two color i2c OLED for my projects. That translates to a display area of 8 lines by 16 or 32 characters depending on the size of the font, taking as reference 8x8 font boxes. The first two lines are yellow and the rest are blue. They look really nice. Again they cost less than 3 Euro on the ebay.
You can see it in the picture together with the cabling harness for connecting it to the APRS board.



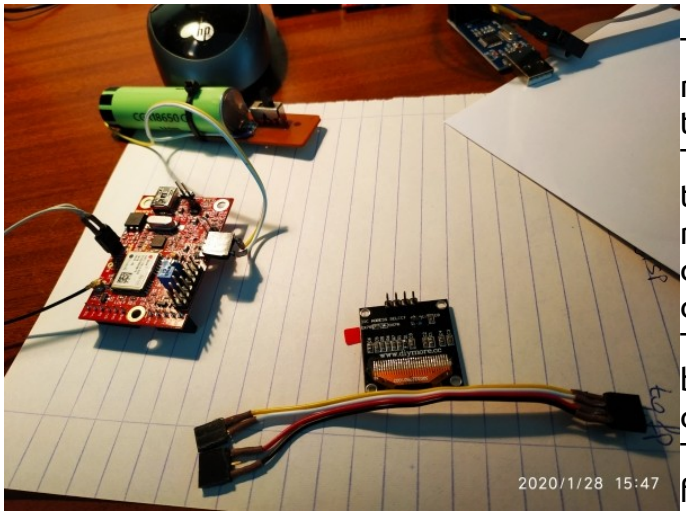
By the way, I made a draft image documentation of the APRS board, so as to avoid mistakes and configure it fast and reliably.



You can easily see where the OLED connects with the two headers, one 1x2 and one 1x3, as well as the LIPO battery 1x3 header. On the 1x3 headers, the middle pin remains unconnected.



This is another useful diagram for the MCU used on the APRS board. It can help you sort out possible questions about the MCU's actual pin numbering which is different from the classic and more common 28-pin D.I.L. pinout of the ATMega328P/PU.

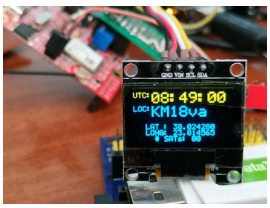


The picture on the left shows all the necessary elements/components to build this first real project.
The OLED has printed on its front side the names of the 4 prins. From left to right, they are: GND (black cable), VIN (red cable), SCL (white cable), SDA (yellow cable). So you cannot go wrong.
The LIPO battery connector from top to bottom is +V (yellow cable), -V (white cable) through an ON/OFF slider switch.
The Push Button is not really necessary for this current applican, so it can remain unconnected.

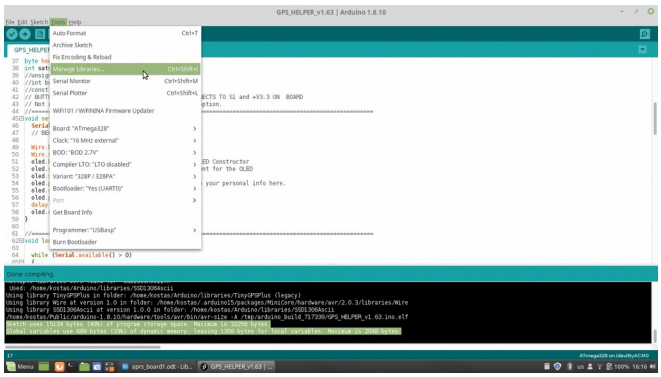
The sketch for this application is GPS_HELPER_v1.63.ino and can be downloaded from SV1AFN's site. Documentation has been as explicit as possible.
For the sketch to compile you need to have installed on the IDE two extra libraries.

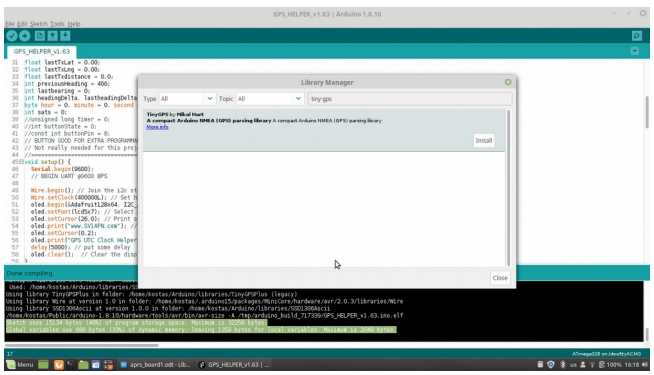
<TinyGPS++.h> //https://github.com/mikalhart/TinyGPSPlus
<SSD1306Ascii.h> //https://github.com/greiman/SSD1306Ascii

The first one is for the GPS and the second for the OLED. The rest of the needed libraries are standard Arduino libraries included/existing on the IDE itself. A Text only fully functional OLED (SSD1306) library that occupies the smallest possible Flash memory space was selected and the use of a single 5x7 font for the same reason. Sketch uses 15134 bytes (46%) of program storage space. Maximum is 32256 bytes. Global variables use 689 bytes (33%) of dynamic memory, leaving 1359 bytes for local variables. Maximum is 2048 bytes. So there is space for more experimentation. The display looks like this:



When I personally install libraries, I prefer to do so by downloading the needed libraries directly from the internet through the Library Manager, i.e. selecting: **Tools>Library** Manager and searching for the library by typing the first few characters of the name. This method allows me to be notified when newer versions of the installed libraries occur.





Remember to upload the sketch by using: **Upload using programmer**

Before doing that, always verify that the code compiles without producing any errors by selecting: **Sketch>Verify/Compile** or by simply clicking the **Check mark** at the top left side of the IDE!

When the sketch has been sucessfully uploaded on the APRS board, it will start running right away. Power it off. Remove the programmer from the board's ISP connector and connect the OLED display as explained above. Power again the board from the battery. The display will show its initializing screen and then will move to the main screen. Initially it will contain only zeroes and the string "JJ00aa" as Locator (no actual data) as the GPS chip needs 3 minutes to get a "Fix" from the Satelite system with a minimum of 3 Sats received. The Helper will not start working if the condition of the minimum number of 3 Sats becomes valid.
An external GPS antenna must be connected to the APRS board for reception. Some data like UTC time may come before the 3 minute interval expires. If the power is switched off and you switch the board again later or the next day, the same 3 minute interval will be needed again.
For this kind of receive only project, an Amateur liscence is not needed. So every DIYer can have it built.

As an epilogue I would like to say that this first code is a mere example to test some of the board's capabilities and obviously good food for thought.
Next steps should be reading the State of Charge of the battery and turning the Helper into an APRS tracker. Help and code contribution from fellow DIYers could speed things up. Being simply alone at my age cannot speed things up!
I am personally satisfied with the Helper function, at least for the time being. I always wanted such a small and portable toy to show me UTC time and the Locator of where I am.
Please do not tell me that I can do that with my Smart Phone and some apps, that I am aware of and have tried before. I was simply after another useful DIY open project.

73, de Konstantinos, SV1ONW.