FT-817 Buddy

Design Notes

Rev 4

I offer the FT-817 Buddy project as a DIY project. The PCB files, BOM, software and build instructions are provided such that you should be able to recreate the current version. Below are some rather lengthy notes on how I've put this together and some of the choices I've made. Hopefully this will enable you to either evolve the design to meet your own particular needs (and maybe contribute back to the project) or just learn about how something like this can be produced using free software, free Internet tutorials and then apply the principles to your own projects.

Basic principle

The FT-817 Buddy is something I have created to solve a couple of problems I have with my FT-817 when I'm outdoors operating portable. Firstly I need more soft-keys - A,B,C are never enough and I hate having to move around the menus for my most used functions. Secondly I usually place the radio on the ground next to me and the display on the radio can be difficult for me to read and see where I'm tuning around the band.

A typical portable situation with my FT-817. No picnic tables here so the radio is on the ground next to where I sit.

My solution to these two problems is to use an Arduino, some push switches and a small LCD display to give me some extra soft-keys and an auxiliary display that can be placed more conveniently (for me) on top of the radio. The Arduino communicates with the radio via the serial (CAT) port and polls the radio for status, reads status items and sends commands to the radio. The official CAT command list from Yaesu is surprisingly limited but Clint Turner KA7OEI has pulled together an amazing amount of work to document how to interact with the radio's EEPROM and monitor/control just about any function or setting the radio has (http://www.ka7oei.com/ft817_meow.html). Note there are some risks involved with interacting with the EEPROM so heed the warnings and note down your individual radio's factory settings in case you need them to restore the radio.

Anyway, back to the Buddy, it's basically just an Arduino, some buttons and a display, connected to the radio's serial port. Pretty simple... but powerful. You could write code for the Arduino to do lots of different things, combine functions into macros, display whatever things you wish to monitor (within the limits of what's available on the CAT interface and EEPROM). You can modify or write your own software to make the Buddy do what you want. Hopefully the functions we have implemented provide good examples of how to use the *ft817* Arduino library (written by Pavel Milanes Costa CO7WT, based on the work of James Buck VE3BUX) to monitor and control the items listed by KA7OEI.

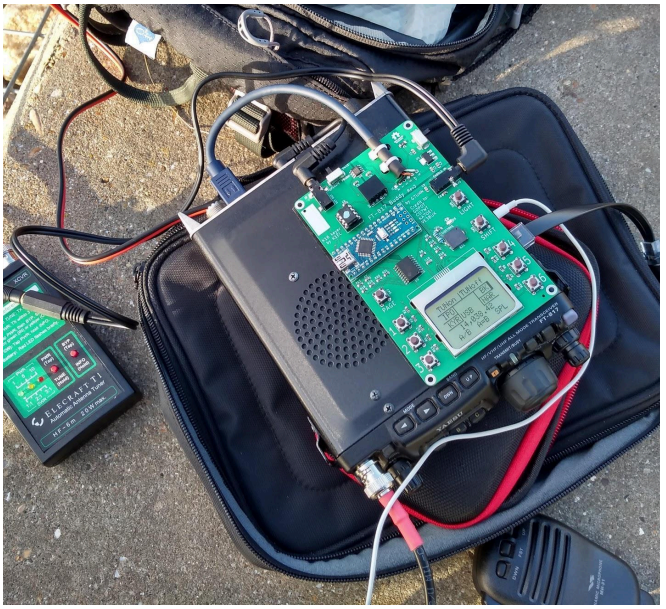
A new feature incorporated in Rev3 is an external CW keyer. Thomas Witherspoon K4SWL / M0CYI seeded the idea for a memory keyer that the FT-817 lacks and it proved relatively simple to graft on one of K1EL's K16 keyer chips and control it from the Buddy's soft-keys. Nice. So now the Buddy is an Arduino, some buttons, a display... and a CW keyer.

It's been a lot of fun learning about PCBs and a fun challenge to put this idea together in a way that hopefully others will enjoy.

Hardware choices

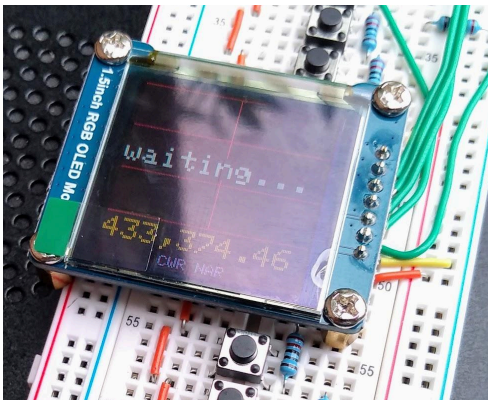
Form factor

Right from the start I had "slim" in mind for the Buddy. I didn't want to add much to my portable setup, just enhance it with a few extra buttons. When I got the display elements I needed duplicated onto the Buddy's LCD I started using the radio in a top-down way which suits my normal backpack-portable operations. I figure there are basically two kinds of radio setups we use as portable operators, they might be described as "with table" or "without table". With my FT-817 I am a "without table" operator (see photo above) and so the top-down view of the Buddy works well for me. Once I'm set up I don't need to look at the front of the FT-817.



Display

I started this project thinking I'd use a small OLED display. They're cheap and low on power draw. I tried a couple, the 128x128 RGB one I used looked amazing indoors but the limitation became apparent outdoors. In direct sunlight (where this device will normally be used) it wasn't up to the job.



I thought about ePaper/ eInk as a really low power option… it would be great for just labelling the soft-keys but by this stage I'd become hooked on the idea of having a frequency display to help me tune around and that needs a refresh rate faster than I think ePaper can deal with.

This led me to the Nokia 5110-style display. These things are widely available and fairly cheap. It's about the right size, it doesn't draw much power, it's perfectly readable in direct sunlight. The 84x48 resolution is a bit limiting but wouldn't be fun if it was easy.

From the Rev2 PCB onwards we take the LCD from a donor module and clip the LCD directly onto our board. That means the LCD module you buy as a donor has to match the footprint we've used on the Buddy PCB. That footprint came from https://hackaday.io/project/25666-yet-another-nokia-5110-lcd-breakout-board and the Sparkfun and Adafruit display modules appear to match (I'm using a Sparkfun display module https://www.sparkfun.com/products/10168 as the donor board, easily available from UK suppliers). On the Rev3 PCB I have extended the footprint to accommodate the slightly larger AZDelivery LCD. We've also discovered a third type of display, the DEVMO 8448 sold on Amazon.com in the US that is very similar to the AZDelivery display… tests pending on this one. The placement of the clips is the same and the contacts all line up so Rev4 is compatible with these 5110-style displays from:

- Sparkfun (tested OK);
- Adafruit (should be the same as Sparkfun);
- AZDelivery (tested OK);
- DEVMO (??) Update when we know what works.

There are some interesting notes on reliability and the pressure-fit of these 5110-style display modules here: https://www.bigmessowires.com/2018/02/22/quest-for-a-decent-lcd/ ...takeaway = if you've fitted the display and you've got problems, try adjusting the clips to make the pressure-fit reliable. The Sparkfun and

Adafruit displays use a conductive rubber insert for the electrical contact and the AZDelivery and DEVMO displays use more traditional metal spring contacts. I don't know which is more reliable.

These Nokia displays run off 3V3 signals and I use resistors between the Arduino and LCD as described in this tutorial. That might be a rubbish way to do it (?) but it seems to work for now.

Microcontroller (Arduino for now)

I started out with an Arduino Nano, mainly because it's cheap, easy to interface to and I had one spare in the workshop. Rev2 used an Arduino Pro Mini but the power draw turned out to be identical to the Arduino Nano (despite what the Internet wisdom suggested) and it complicated the programming so Rev3 is back to the trusty Arduino Nano.

It's useful that the FT-817 serial interface operates at 5V levels so you can work directly with a 5V microcontroller. It's also the reason I'm sticking with 5V Arduino versions.

Experience with using a software serial port wasn't great (intermittent glitches on reading data and possibly increasing the chance of a write error screwing up the EEPROM) so Rev2 onwards use the hardware serial port (pins 0,1) to communicate with the radio. The Operate/Program switch on the PCB breaks the RXD line to the radio when you're programming the Arduino. It means we can't run a serial monitor on a PC connected to the Arduino while it's working with the radio but we do have a display on the PCB so debugging print statements can be sent to that instead if needed.

I tried an Arduino Nano Every as an alternative to the Nano because it decouples the USB port from the hardware serial lines and has more memory allowing me to continue writing lazy inefficient code… However it's not as common as the Nano, has a confusing name, is more expensive, needed the timer functions rewriting for the different processor and draws an extra 20mA over the Nano. Unacceptable for this application.

I've looked at the Teensy line of microcontrollers… they look exciting and would allow simultaneous hardware serial interfaces to both radio and PC (useful for software debugging but not essential to operation)… but the product line is a bit overwhelming and they look a little expensive/ overpowered for this application. Also, with the increase in processor power, I expect the current draw may be higher than we want. And it probably requires some 5V to 3V3 logic level conversion between the microcontroller and the radio. Maybe someone can enlighten me, drop any suggestions over to the Discussion page.

There's also the option to just build an AVR microcontroller directly on the Buddy PCB... but lots of people get freaked out by that kind of thing, using a basic Arduino probably keeps this project within reach of a wider audience.

Buttons

The wiring of the buttons may strike some as unusual or inefficient but there is method in my madness.

The main aim of the Buddy is to provide six soft-keys that can be assigned different functions to drive the FT-817. Therefore it is assumed that these buttons will only be pressed one at a time. It follows then that it's OK to stack these buttons (SW1-SW6) up with a resistor network and feed them all into a single input pin, A0 on the Arduino.

The LIGHT, PAGE and SHIFT buttons soon came along and it made sense that they could be pressed simultaneously with other buttons as key combos (e.g. using the SHIFT key as a guard against accidental transmission). So SW7-SW9 are wired into the Arduino on individual digital pins.

There are plenty of other ways to arrange multiple buttons on Arduinos but I'm happy that this is an efficient approach given the difference in usage between SW1-SW6 and SW7-SW9. As the project has expanded

to include the K16 keyer this approach has left me with enough Arduino pins to drive the four button inputs on the K16.

Use of SMT devices

It's 2020. Any budding home constructor now needs to be OK with handling and using SMT devices in their projects. You can still use through-hole components and modules for experimenting on breadboard but it's now easy to design the PCB using free tools like KiCAD and low-cost manufacturers like JLCPCB will even solder on most of the SMT stuff for you!

Given all this I'm quite happy with the use of SMT devices in this project. They reduce the size of the PCB to something quite neat, JLCPCB can handle most of the work placing the components and people are still left with a couple of SMT devices to solder on themselves and feel like they've achieved something at the end. If you want to go all the way you could easily hand-solder all of the components on the Buddy (I have on Rev1 and Rev2, several times).

The hand-soldering of the 0805 size LEDs is driven by the desire for low-current consumption. 0805 size is what's required to be compatible with the LCD modules we're using and it's fairly easy with a magnifier, "helping hands" or whatever, most people will have something like that in their workshop. At this time, JLCPCB don't have any low power LEDs in their parts library for factory placement so it's up to us to buy them and solder them on ourselves.

The SMT piezo buzzer may be slightly frivolous… I wanted to keep the component height down on the PCB to help the design of a case when the time comes… and given that we have to hand-solder some small LEDs, why not give one of these a go at the same time.

K16 keyer button triggering and speed adjustment

The K16 keyer chip uses a slightly awkward resistor network to allow four button inputs AND a speed potentiometer adjustment onto a single pin on the K16 IC, described in the K16 User Manual. It's clever but to drive these button inputs from the Arduino I have to use transistor switches (unless I added on relays but that seemed like overkill). The transistor switches (courtesy of the Darlington pair "low-side drivers" in the ULN2803 chip, a common device for this kind of thing) introduce slight voltage offsets into the K16 resistor network so I've had to fudge around the resistor values by experimentation on breadboard to make it work. I think I've got it about right and it works for me in testing but if someone comes up with better resistor values then let us all know.

Given the delicate balance of the K16 resistor network I decided not to include a speed pot and instead rely on the K16's "Fast Speed Change Feature" whereby you just press CMD and hold a paddle to increment the CW speed up or down. It works well, it saves having to find room on the PCB for a pot and we don't have a speed knob growing the size of the Buddy.

Isolation components

Proving the concept of the Buddy on prototype board showed that there was some interference from the serial lines into the radio's receive audio, heard as pulsing noises whenever the radio was polled for status a few times every second. The prototype setup was unshielded with wires all over the place. Noise only became apparent when an antenna was connected to the FT-817.

Although the noise was probably radiated rather than conducted, the first measure taken was to add in a couple of isolation components just because it was easy to do on the breadboard:

- B0505S-1W DC to DC isolator for the power line;

- ADUM1201 digital isolator for the tx and rx serial lines. I can't remember quite why I picked this over an opto-isolator… I think it was just that rx and tx were available on a single IC so it made the board easy. I don't regret it.

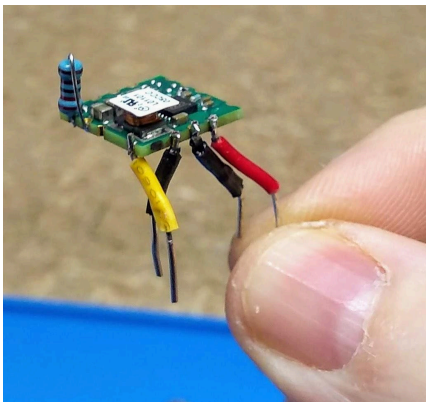
Adding these components to the prototype board did reduce the noise a little bit but I guessed the majority of the interference was radiated not conducted so I crossed my fingers and hoped that moving to a PCB with short serial lines surrounded by ground plane would cure it. Because I'm a little paranoid about noise (this is a device we're connecting to our receiver's power supply) I stuck with the digital and DC isolator components and designed the PCB to have electrical isolation between the FT-817 and the Arduino, it seemed like a good idea and a fun challenge for the PCB layout (again, it wouldn't be fun if it was easy, right?).

So, the Rev3 PCB retains these isolation components. Maybe they're not necessary. Maybe they could be removed to save some cost and shrink the PCB a little bit. I don't know because I haven't tried yet. A couple of jumper wires across the PCB in place of these components would probably tell us... if you try it, let us know!

Voltage regulators and current draw

Yes, the Buddy uses a simple linear voltage regulator to drop the "13.8V" input voltage from the radio[1] to the 5V required for the Arduino. Yes, it's very inefficient and probably responsible for 50% of the power consumption of the Buddy. Yes, I did lose sleep over this...

The solution seemed to be either to design a low-noise switch mode power supply or add a pre-made module. I searched around a bit and tried a dinky little Murata OKL-T/1-W12P-C SMPS module (see photo). It has a switching frequency of 800 kHz which probably keeps it clear from most HF bands. I tried it on prototype board but, surprisingly, it only offered a small saving on the power draw that the linear regulator was showing (17% less than the linear reg, about 5mA saving). Adding in the Murata module was going to significantly add to the shock factor of people unfamiliar with soldering SMTs so I decided the juice wasn't worth the squeeze. We stick with the simple linear regulator for now.



Current draw of the Rev3 design is less than 50mA @ 12V from the radio. Measurements were:

- Idle (Arduino running, polling Freq/Mode), backlight off - 43mA
- Idle (Arduino running, polling Freq/Mode), backlight on - 46mA
- Keying, buzzing, backlight on - 49mA

The K1EL K16 chip only contributes about 2mA of the current draw when idle. When keying the current rises about 5mA.

For now I think I can live with those figures and the peace of mind that linear regulators bring but if you have the electrical chops to help us significantly reduce the power draw with a SMPS (or any other ideas) without increasing complexity too much, please get in touch.

I haven't tried any sleep mode trickery on the Arduino, my guess is that would be incompatible with our usage which probably needs the Arduino awake and available to do stuff most of the time. But if you're a whizz on sleep mode give it a try and let us know the results.

[1] Input voltage from the radio is between 9V and 14V despite Yaesu's highly misleading "13.8V" label on the pinout

What can the FT-817 supply from the ACC port? Good question. There doesn't seem to be an official figure for this. There is a 10 Ohm current-limiting resistor on the voltage supply line of the FT-817ND (older serials may not have this) and the power rating of that tiny resistor seems to limit the supply to 80mA before it goes pop. The BHI "CAT-MATE" accessory user manual seems to be the most definitive thing I've seen regarding current draw from the ACC port. That document states there is a 60mA limit on current draw from the FT-817 ACC port and that seems to line up with the 80mA rating of the resistor with a bit of safety margin. The Rev3 Buddy design appears to stay below 50mA through all of my testing.

If you're not comfortable with powering the Buddy direct from the FT-817 then I have provided an extra ground pad next to J1 to make it easy to power it from a separate cable, see the diagram in the Build Instructions.

PCB design

Not much to say here, I did the hardware design for this project in KiCAD which I really enjoy using. I've just learned how to do it with a couple of Internet tutorials (mainly this one) and Google. So much easier than Eagle, it doesn't carry around all the baggage that Eagle seems to have. Maybe the component libraries aren't quite as extensive as Eagle yet but I've not had any problems finding the parts I needed to import out there on the Internet. Check it out. Lot's of great tutorials out there.

Software design

I use the word "design" very loosely here! My software technique is generally limited to Googling things, copy/pasting code snippets and then Googling the error messages until I force it to work. Fortunately, placing the code on GitHub allows us to improve it collaboratively. If you want to get stuck into that, drop us a line on the GitHub Discussion or do a fork and a Pull Request or something and I'll hook you up (but bear in mind I'm a beginner on all things git/GitHub so you might need to hold my hand!).

I've tried to use descriptive variable names and comment the code throughout what I have written. The basic structure of the code is a little clunky but it works and I hope others may be able to improve on it.

Major kudos to Pavel CO7WT who wrote the ft817 library that allows the inner workings of the FT-817 serial interface to be abstracted away from the more basic structure of driving the Buddy display and buttons triggering commands mainly from the library. Pavel actually wrote the library 'blind' - he doesn't have an FT-817! Amazing work. On a side-note, if anyone knows anyone willing to hand-carry an FT-817 into Cuba as a legal import for him, please reach out!

One of the major items on the to-do list is to tidy up the code at the front end of the sketch and make it easy for people to customise the soft-keys on each page to their preferences. It would be nice to have some kind of drag-and-drop web application that could be used to layout the soft-keys as you wish and then generate the corresponding code segments to be copy/pasted into the sketch… but now I'm dreaming.

The functions that have been implemented so far probably cover the majority of structures in the ft817 library so should serve as good examples when people want to customise the code for new functions. So far these are:

- Basic CAT commands, mainly from VE3BUX's original FT857D library
- EEPROM reads and writes to toggle "global" parameters e.g. BreakIn on/off, Keyer on/off
- EEPROM reads and writes to toggle VFO/band-specific parameters e.g. NARrow filter on/off, IPO on/off
- EEPROM writes to an address with some payload data e.g. Keyer speed = <integer value>

A little bit of reading between the main sketch, the library code and KA7OEI's interface description should explain what's going on when you want to get stuck in.

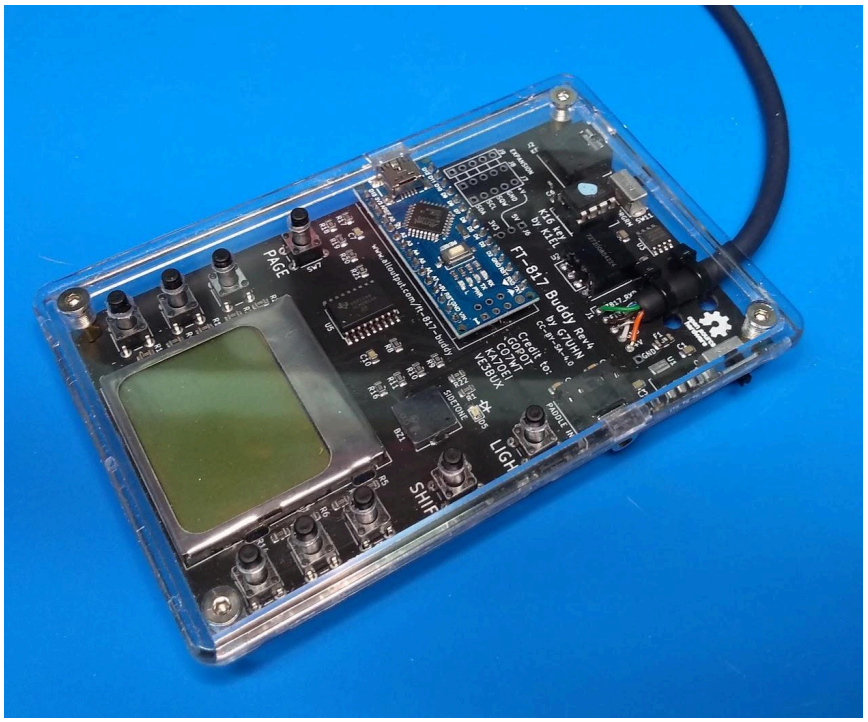
Use of Arduino IDE vs. Visual Studio Code/ PlatformIO

There's enough code here with the library to make the limitations of Arduino IDE obvious… Pavel pointed me at Visual Studio Code and the PlatformIO plugin for Arduino and it's a massive step forward from working in Arduino IDE. Spending some time getting to know VSC and PlatformIO IDE was well worth the effort and makes it much easier to move around the code and the ft817 library (e.g. right-click on a function name to go to the function definition, etc.). Actually it makes working on the code quite easy and it links to GitHub from within the interface.

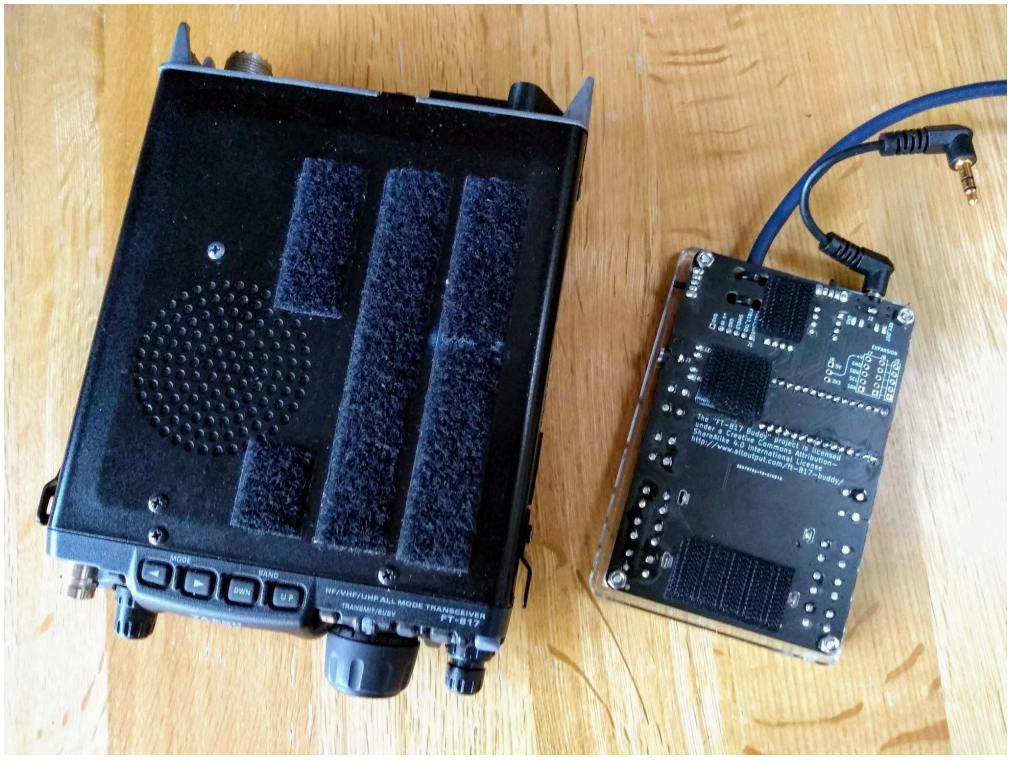
Using PlatformIO does introduce some slightly awkward moments when the code is to be loaded in Arduino IDE (like renaming the main program and removing the Arduino.h include…) but this is the way it is until I find some button that auto-magically converts the PlatformIO format into Arduino IDE format before uploading to GitHub.

Where's the case?

Work is in progress on an enclosure based on a Hammond 1591TC clear lid. The Rev4 PCB is sized to be fitted to this lid and I'll work up a 3D-print design for the lower half of the case. It will need a bit of drilling and filing but seems like a good use of an off the shelf component and saves you having to cut a square hole in something for the LCD. See Enclosure Notes for more details.



You can use velcro to stick the PCB on top of the radio if that's what you want. Don't go industrial on the velcro (e.g. 3M dual lock) you might break the PCB when it's time to detach from the radio. Don't ask me how I know that. A couple of small squares of standard Velcro will be fine.



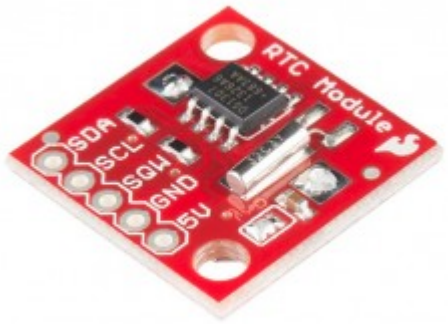
Expansion Header

The idea for the expansion header comes from some blatant scope creep, a fanciful desire for a clock on the Buddy. Yeah, I know we're all wearing a watch or have a phone but it always seems your watch is buried under your sleeve while you're logging that rare DX… The two ideas considered here are either

A. Real-Time Clock (RTC) chip... which would also require a coin-cell battery and a bunch of Arduino code to program the clock on initial setup;
B. GPS/GNSS module which would supply accurate time shortly after power up and also provide Lat/Long and/or Maidenhead grid locator to the user but will probably pull a fair amount of current when it's running…

There is also the very mundane Option C which is just to velcro a small clock to the top of my radio, but where would be the fun in that? Anyway, recognising that this is growing the project a bit, both the RTC and the GPS idea could be added on as an optional expansion module. Both options would want a voltage/ground and a serial interface to the Buddy's Arduino so the Expansion Header adds this hardware interface to the Buddy PCB and allows for this to be explored or ignored.

I'm currently using the Sparkfun DS1307 RTC module (part number BOB-12708) to run the RTC on my Buddy and it fits very nicely on top of the Rev4 PCB. It's available from a range of suppliers in the UK, US and presumably around the world. Open source design too.


RTC Module
SDA
SCL
SQW
GND
5V