Creality 4.2.2 and 4.2.7 mainboard marlin TMC2208/TMC2225 uart mod
by LookAtTheShinyShiny

DISCLAIMER
All modifications are done at your own risk! To the best of my knowledge, the information in this document is correct. If you see any errors, feel free to let me/people know. This document is given freely, do what you like with it.

This mod is a continuation of the original 4.2.2/ Linear Advance TMC2208 UART mod and builds on the work by Wong Sy Ming. A big thank you to the author!!

I hope it will fill in some gaps that I found in the original documentation as well as fill in all of the details necessary to perform this mod on a 4.2.2 and a 4.2.7 board.

The Creality™ 4.2.7 PCB is the retail version of the 4.2.2. board, but is a slightly different design than the 4.2.2 boards that are stock in the Ender™ 3 v2 printers.

This mod is designed around my own setup, which includes a BLTouch™ from Acculabs, so I'm not using the Z-endstop pin. I'm also not using a filament runout sensor. If you're using either the Z-endstop or the filament runout, then you can still do the Linear Advance mod for the extruder motor, or if you really want control of all 4 motors, I have listed in the document some extra pins on the STM32F103 chip that should be usable.

4.2.2 vs 4.2.7 differences

The 4.2.7 board uses a TMC2225 chip which is a physically different footprint to the TMC2208. This is useful in a few ways:

1. It's easier to solder to the pins if necessary.

2. The TMC2225 has a few different pins/features. If you just want Linear Advance on the extruder, there's a spread pin that's tied low (stealthchop mode) which can be connected to the appropriate voltage supply, although this would need to be confirmed as a method of achieving Linear Advance.

3. The TMC2225 uses MS1 and MS2 to configure the standard microsteps from 4 to 32. This is an improvement on the TMC2208 which only gives us 1 to 16. Again, just simply tying the 2 pins high would double/quadruple the resolution of the print moves. This assumes that microplyer with 256 microstep interpolation isn't already on?

4. TMC UART allows easy access to control the motor current of the TMC2225 for each motor

5. Of course if you have access to the pdn_uart pin then you get access to all of the above in a single pin :-) The other advantage of UART is that you gain independent control of the EN pin for each motor, which does away with the need for the EN pin and reclaims a pin for other uses. It also allows us independent control of when each motor is enabled.

This mod makes the PDN_UART work in half-duplex mode on the STM32 side of things (TMC chips already implement their own half-duplex mode). We get away with using a single pin for TX and RX.

The mod also makes use of the Arduino based Software Serial Libraries, which makes it much easier to implement.. If we had to use hardware serial pins, we'd need twice as many and they'd probably have to be specific pins, most likely already in use elsewhere on the board.

Things you will need

A soldering iron, preferably with a fine tip
Solder, the thinner the better to assist controlled flow and avoid bridging
Solder wick (optional)

An IDE, VSCode/VSCodium with platformio installed
A fresh copy of the Marlin 2.0.X (or bugfix branch) Github repo

Useful Resources

For reference of trinamic specs and data sheets:
https://www.trinamic.com/products/integrated-circuits/details/tmc2208-la/

https://www.trinamic.com/products/integrated-circuits/details/tmc2225-sa/

Here is a list of the recommended Vref voltages for each motor, they are different between each axis!!
https://support.th3dstudio.com/hc/product-information/3rd-party-control-boards/creality-boards/creality-v4-2-x-tmc-boards-recommended-vref/

On its own, this information is useful to anyone working in standalone mode as it allows you to set the Vref without really knowing what current the motor will draw, but to be useful in TMC UART mode, by default, TMC UART mode does not use the Vref pot, so we need to calculate the current we actually want, to be able to set it correctly. The following web page contains a calculator:
https://wiki.fysetc.com/TMC2208/#calculator

It only works on current, so you have to set the current you think you need to get the Vref value. Hit calculate and see whether it matches your desired Vref. If it doesn't, adjust the current value accordingly until you get a Vref match. Just in case you're reading this at 4am, or just don't want to do it yourself, I have provided the following current ranges according to the given voltage range for each axis.

Table.1

AXIS	VREF (IN VOLTS)	CURRENT (IN MA)
X	**1.10-1.20V**	**780-850ma**
Y	**1.00-1.10V**	**710-780ma**
Z	**1.15-1.20V**	**815-850ma**
E	**1.35-1.40V**	**960-990ma**

These values seem to be very specific, but also quite vague as there is no context around why these values are desired. Although we’re told adjusting the Vref will allow us to avoid lost steps and reduce heat, there doesn't appear to be any information on how much heat is actually bad for the steppers. The X, Y and Z motors are all the same spec, so they could in theory all be run at the

same Vref/current. It would be really useful to know if there is a reason why they aren't run like that by default?

Why do we need these values?

This information is very useful whether you use a 4.2.2 board with TMC2208 driver chips or the 4.2.7 with TMC2225, while running a TMC UART enabled firmware, because the motor current can be set directly using Gcode from your favourite printer console app (Octoprint, Pronterface, Repetier etc.).

We can also compile the current values directly into the firmware. As it stands, all current values are set to 800ma in the firmware, which is in the range of the X motor current draw but not for Y, Z and E motor current draw. We definitely need to change that!

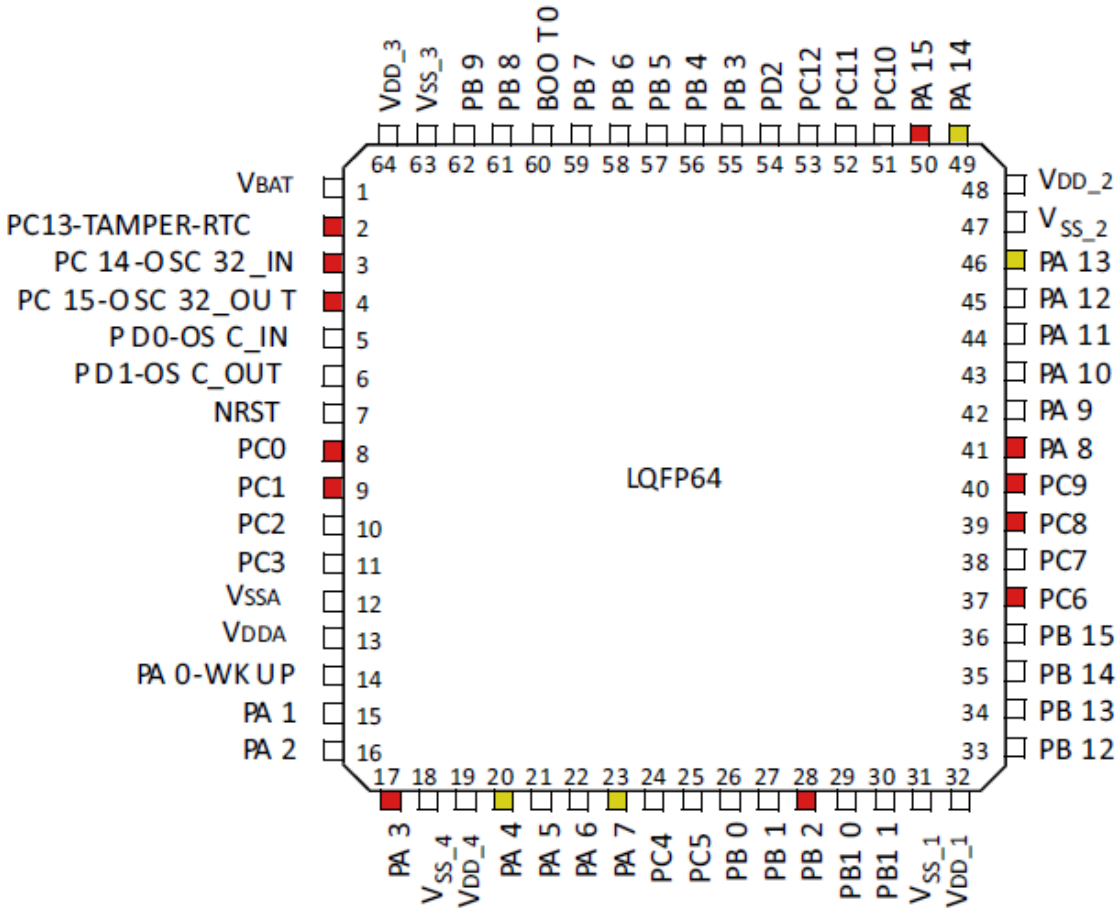
How to gain access to TMC UART on XYZ

The TMC UART mode works in various ways. The first is independently with all motor drivers requiring their own pin on the STM32 to connect to the PDN_UART pin on each driver chip. The second way is to connect all of the PDN_UART pins to a single pin on the STM32, but that only gives us write access, so we'll be concentrating on the former method.

3 more pins

To achieve this, we will need 3 more pins, one for each of the X, Y, Z motor drivers. We could do the same as the original 4.2.2 Linear Advance mod and solder directly to the STM32 chip, as there are around 16 free pins (PC0, PC1, PC6, PC8, PC9, PC13, PC14, PC15, PA3, PA4, PA7, PA8, PA13, PA14, PA15, PB2 (see pins marked red in **fig.1** below.) available and whilst this method would be achievable, no one really wants to be soldering directly to the MCU, so I have another method...

Fig.1



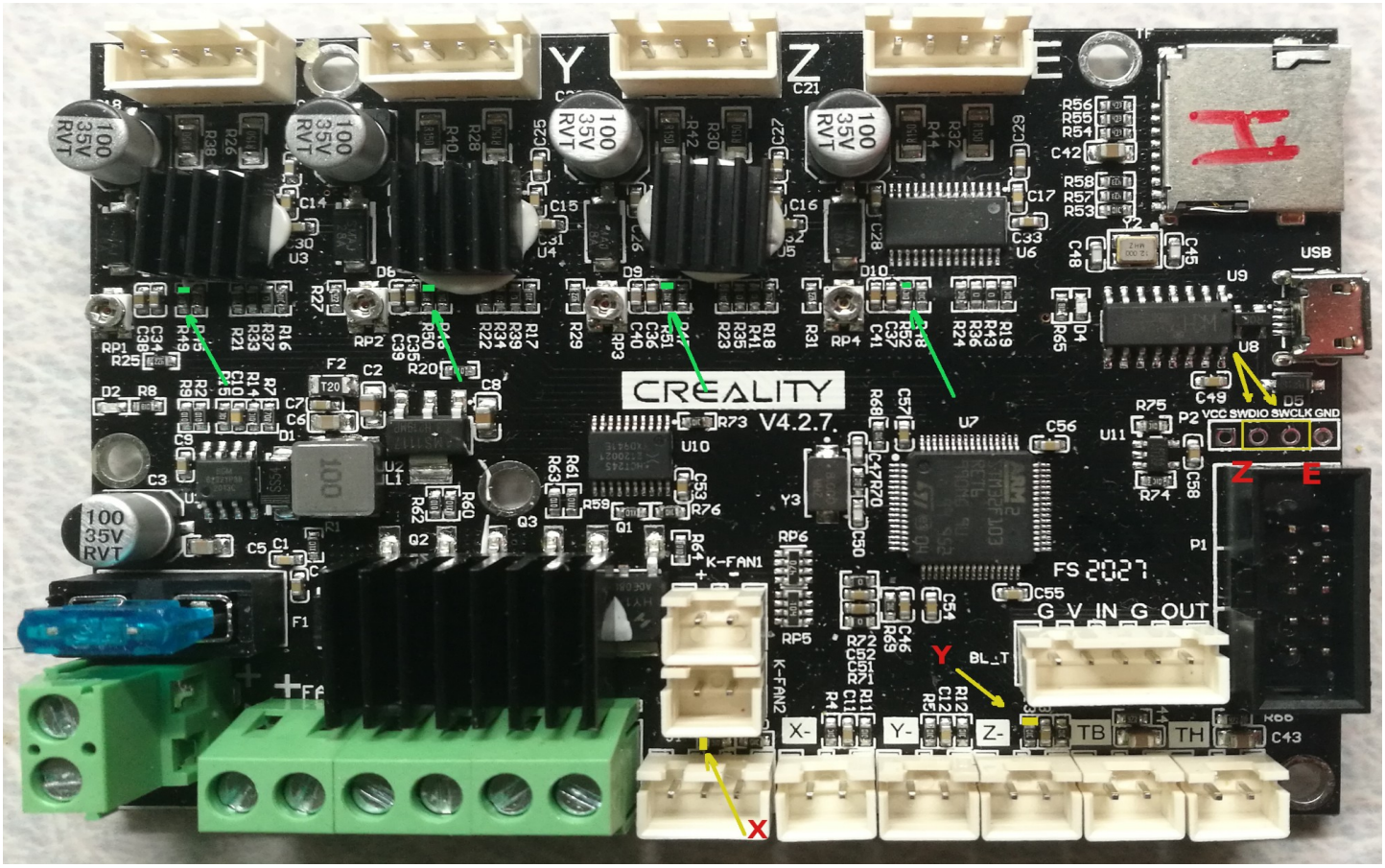


Fig.2

Whilst figuring out the rough board design of the 4.2.2, I noticed that there's an unused port marked VGS next to the endstop ports. It turns out that this port is used for a filament runout sensor and uses pin PA4. I'm also using a BLTouch™ in my setup, so that frees up the Z endstop pin PA7. Lastly just above the LCD connector is a 4 pin header (populated with pins on the 4.2.2, unpopulated on the 4.2.7 board). This has 2 I/O pins, SWDIO and SWCLK which are connected to pins PA13 and PA14 respectively on the STM32 chip. So that's 4 pins we have direct access to without needing to solder directly to the STM32 chip at all. Happy Days!! (see pins/solder points marked yellow in fig.1 and fig.2)

I should take the time to mention here that PA13 and PA14 are directly connected to the STM32 chip. There are no external resistors tying the pins high or low, but the PA4 and PA7 pins may have pullup/pulldown resistors attached as they are input pins where a known state is required to avoid false readings from those pins.

Obviously, if you're using the filament runout sensor and Z endstop then you'll need to figure out alternative pins (I will list those at some point) or you can use a Raspberry Pi and Octoprint to monitor the filament runout sensor, at the time of writing I'm not sure if this could also be done for the Z-endstop.

I am also wondering whether it would be possible to connect all of the endstop switches to the same pin with careful homing methods? This would free up 2 more pins which would give us 3 'spare' pins if you include the EN pin, which in turn would allow us to add a 5th motor + driver, for dual Z motors with independent control perhaps?

How to Mod the board (TL:DR)

The mod is achieved in 2 parts. Part 1 is soldering to the correct points on the board. Part 2 is editing and compiling our own firmware. The following pins should also be available on the 4.2.2 boards!

1. Soldering - Let’s get the electric glue in the right place...

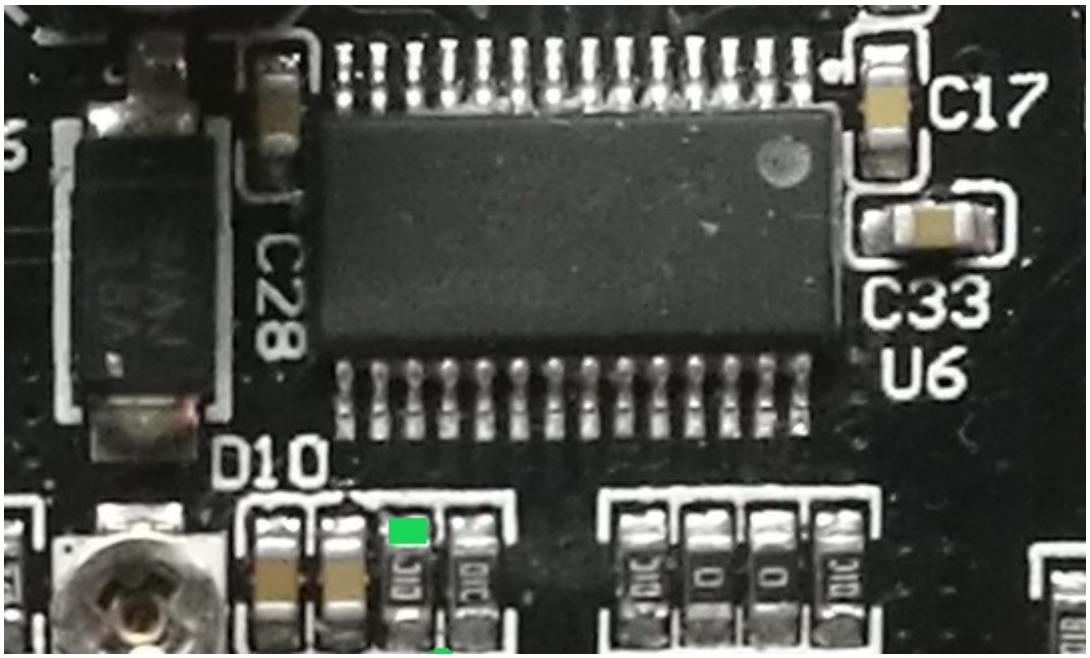
It's up to you which of the 4 pins you connect to each motor but it's probably easiest to do the following setup for the 4.2.7 board:
X PDN_UART - R49 - PA4
Y PDN_UART - R50 - PA7
Z PDN_UART - R51 - PA13
E PDN_UART - R52 - PA14

And for the 4.2.2 board:
X PDN_UART - R16 - PA4
Y PDN_UART - R17 - PA7
Z PDN_UART - R18 - PA13
E PDN_UART - R19 - PA14

(Please NOTE!!! - When soldering to the resistors, always solder to the TMC2225/TMC2208 side of the resistor)

On the 4.2.7 boards, the PDN_UART pin is tied to the DIAG pin, the DIAG pin is output only and indicates an error when it reads high. Both pins are tied to ground via a resistor. I don’t know if this will impact the mod, but if it does then it should be straightforward enough to cut the DIAG pin trace, as all diagnostics will be done on the chip via UART. Furthermore, we don't actually check the DIAG pin for errors, so it should be safe to cut.

Fig.3

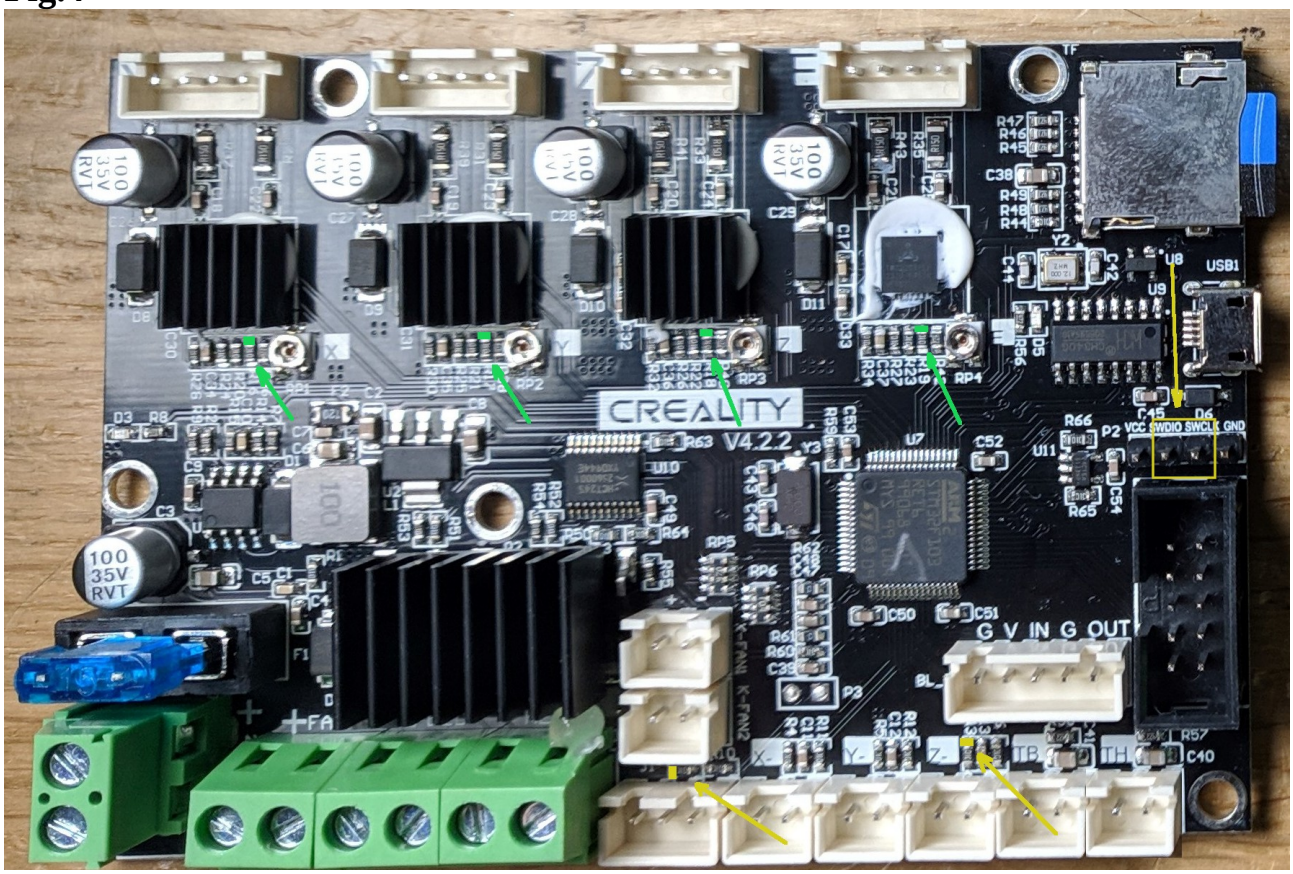


If you look at the picture of the TMC2225 chip layout (fig.3 above) on the 4.2.7 PCB and compare it to your board (your SD card slot should be top right as you look at it) just to the right of the VREF ADJ potentiometer are 4 SMD parts, 2 caps and 2 resistors. We will be soldering to the 1st resistor (R52, R51, R50, R49) in that row. This will be the same place for each of the 4 motor driver chips. For your reference, the PDN_UART pin is pin 17 on the TMC2225 and is the 3rd pin from the left as you are looking at the chip.

You might need to remove the heat sinks and clean up the thermal glue to get space to solder. If you do, you will need to use thermal glue or heat sinks with pre-applied pads to reapply the heat sinks once the mod is completed. The thermal glue can be cleaned off with a toothpick to pick off the main bits and some Isopropyl Alcohol (IPA) to clean up the residue.

That's roughly it for prep, now just solder each PDN_UART pin to the pins shown in the pictures below for each board. (fig.4 for the 4.2.2 board and fig.5 for the 4.2.7 board)

Fig.4



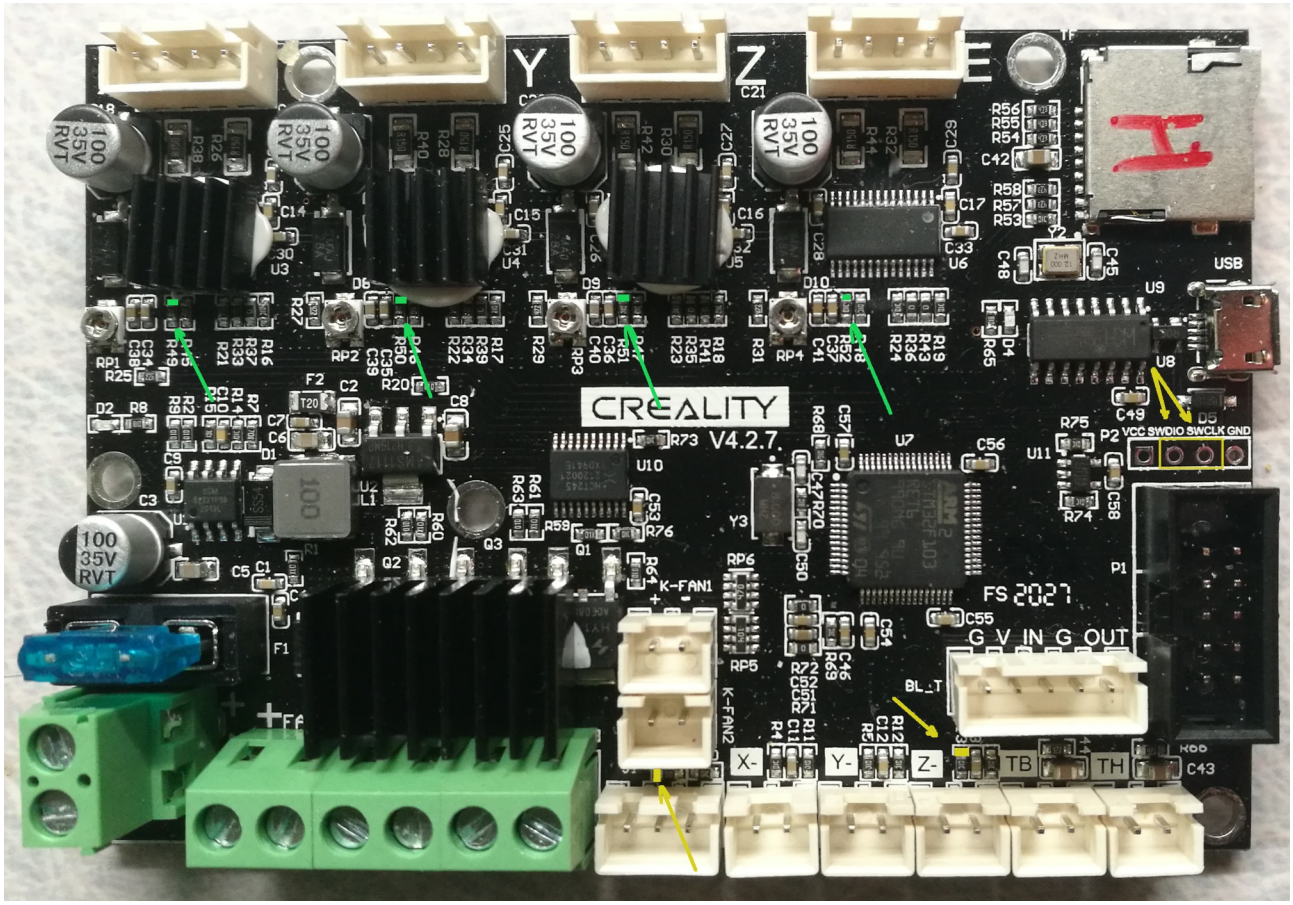


Fig.5

2. Editing the firmware, so we keep the magic blue smoke in the right place...

Now that the hardware mod is done, we can edit the firmware. We will be editing 3 files from the Marlin firmware, Configuration.h, Configuration_adv.h and pins_CREALITY_V4.h. You might be wondering why we're not editing pins_CREALITY_V427.h? The answer is simple - pins_CREALITY_V427.h only contains the changes between 4.2.2 and 4.2.7 boards, which is just the swapped STEP and DIR pin assignments for each axis. Then it includes the pins_CREALITY_V4.h file, so in essence everything we need to adjust for the 4.2.7 is contained in the pins_CREALITY_V4.h file!!

Edit the following files:

pins_CREALITY_V4.h

Search for Z_STOP_PIN and comment it out:
// #define Z_STOP_PIN PA7

Now do the same for FIL_RUNOUT_PIN:
// #define FIL_RUNOUT_PIN PA4

Now Scroll to the end of the file and add the following:

// Added this section for TMC2208 uart control - Requires hardware modification
#if HAS_TMC_UART

#define X_SERIAL_TX_PIN PA4 // Reuse Filament runout sensor pin
#define X_SERIAL_RX_PIN PA4
#define Y_SERIAL_TX_PIN PA7 // Reuse Z end stop pin
#define Y_SERIAL_RX_PIN PA7
#define Z_SERIAL_TX_PIN PA13 // Reuse SWDIO 'debug' header pin next to LCD socket, unpopulated on 4.2.7 boards
#define Z_SERIAL_RX_PIN PA13
#define E0_SERIAL_TX_PIN PA14 // Reuse SCLK 'debug' header pin next to LCD socket, unpopulated on 4.2.7 boards
#define E0_SERIAL_RX_PIN PA14
#define TMC_BAUD_RATE 19600
#endif

Next file:
Configuration.h

Search for USE_ZMIN_PLUG and comment it out:
//#define USE_ZMIN_PLUG

Now search for FILAMENT_RUNOUT_SENSOR and comment that out too:
// #define FILAMENT_RUNOUT_SENSOR

Search for X_DRIVER_TYPE and change the X,Y,Z and E driver type:
#define X_DRIVER_TYPE TMC2208
#define Y_DRIVER_TYPE TMC2208
#define Z_DRIVER_TYPE TMC2208
#define E0_DRIVER_TYPE TMC2208

Now search for #define S_CURVE_ACCELERATION and comment it out:
#define S_CURVE_ACCELERATION

Last file:
Configuration_adv.h

Search for STEALTHCHOP_E and comment it out. You can leave the defines for Stealthchop XY and Z. Changing to spread cycle is for the Linear Advanced mod. I'm not sure if it's immediately useful on the other axis.

#define STEALTHCHOP_XY
#define STEALTHCHOP_Z
// #define STEALTHCHOP_E

Search for //#define LIN_ADVANCE and uncomment it:
#define LIN_ADVANCE

Next look for CHOPPER_TIMING and change it to 24V:
#define CHOPPER_TIMING CHOPPER_DEFAULT_24V // All axes (override below)

Now look for TMC_DEBUG and make sure it's defined, this gives us more detail in M122:
#define TMC_DEBUG

The last part is editing the default settings that we want for our driver chips. If you refer back to the Vref/current table (Table.1) search for #if HAS_TRINAMIC_CONFIG and in that section edit the following:

```
 #if AXIS_IS_TMC(X)
   #define X_CURRENT       850       // (mA) RMS current. Multiply by 1.414 for peak current.

 #if AXIS_IS_TMC(Y)
   #define Y_CURRENT       780
 #if AXIS_IS_TMC(Z)
   #define Z_CURRENT       850

 #if AXIS_IS_TMC(E0)
   #define E0_CURRENT      990
```

Editing done, Time to compile

Compile your firmware and flash it to the board as you would normally. Once you've flashed the unit, it's time to connect everything back up. If you're worried that you might have done something wrong, you can just connect the main 24v power to the board and you can leave the motors unplugged. You will also need to connect the USB socket on your PCB to your choice of console hardware (directly to a USB socket on a PC or a Raspberry Pi). If your PC won't connect via USB, it will probably require the following driver:
http://www.wch-ic.com/downloads/CH341SER_ZIP.html

Once you've reconnected your PCB correctly, it's time to issue some Gcode to to see if your mod has succeeded...

In your printer app console type the following, then hit Enter:

M122

You should get something similar to this returned by your newly minted firmware:

```
Send: M122
Recv:     X  Y  Z  E
Recv: Enabled    false  false  false  false
Recv: Set current  850  780  850  990
Recv: RMS current  826  764  826  939
Recv: MAX current  1165  1077  1165  1324
Recv: Run current  26/31  24/31  26/31  16/31
Recv: Hold current  13/31  12/31  13/31  8/31
Recv: CS actual  13/31  12/31  13/31  8/31
Recv: PWM scale
Recv: vsense    1=.18  1=.18  1=.18  0=.325
Recv: stealthChop  false  false  false  false
Recv: msteps    16  16  16  16
Recv: interp    true  true  true  true
Recv: tstep    max  max  max  max
```

Recv: PWM thresh.
Recv: [mm/s]
Recv: OT prewarn false false false false
Recv: pwm scale sum 15 14 15 10
Recv: pwm scale auto 0 0 0 0
Recv: pwm offset auto 36 36 36 36
Recv: pwm grad auto 14 14 14 14
Recv: off time 0 0 0 0
Recv: blank time 24 24 24 24
Recv: hysteresis
Recv: -end 2 2 2 2
Recv: -start 1 1 1 1
Recv: Stallguard thrs
Recv: uStep count 8 8 8 40
Recv: DRVSTATUS X Y Z E
Recv: sg_result
Recv: stst
Recv: olb
Recv: ola
Recv: s2gb
Recv: s2ga
Recv: otpw
Recv: ot
Recv: 157C
Recv: 150C
Recv: 143C
Recv: 120C
Recv: s2vsa
Recv: s2vsb
Recv: Driver registers:
Recv: X 0x80:0D:00:00
Recv: Y 0x80:0C:00:00
Recv: Z 0x80:0D:00:00
Recv: E 0x80:08:00:00
Recv:
Recv:
Recv: Testing X connection... OK
Recv: Testing Y connection... OK
Recv: Testing Z connection... OK
Recv: Testing E connection... OK
Recv: ok
[...]

With the following being the most important part for our purposes. If you've been successful, the last few lines will look like this:

Recv: Driver registers:
Recv: X 0x80:0D:00:00
Recv: Y 0x80:0C:00:00
Recv: Z 0x80:0D:00:00
Recv: E 0x80:08:00:00
Recv:

Recv:
Recv: Testing X connection... OK
Recv: Testing Y connection... OK
Recv: Testing Z connection... OK
Recv: Testing E connection... OK
Recv: ok

If the Driver registers are all 00 or all FF, there's an error somewhere (could be firmware or your soldering...)
This will be reflected in the feedback from the driver and will look something like this:

X 0x00:00:00:00 Bad response!
Y 0x00:00:00:00 Bad response!
Z 0x00:00:00:00 Bad response!
E 0x00:00:00:00 Bad response!
Testing X connection... Error: All LOW
Testing Y connection... Error: All LOW
Testing Z connection... Error: All LOW
Testing E connection... Error: All LOW

If you get an ALL LOW error, check your soldering and recheck that you've made all of the necessary edits to the firmware and that the firmware compiled correctly. Also make sure that the 24V power has been correctly connected to the PCB. When I first performed the mod, I made the mistake of just testing using power from the USB to Serial connection, so double check your power connections if you get any errors.

Finishing up the mod

There are 3 relevant Gcodes now available:

M122 – TMC Debugging, which gives detailed information about how the unit is performing/setup.

M569 – Set TMC Stepping Mode, which allows you to read the current stepping mode or switch between stealthchop and spread cycle modes.

M906 – TMC Motor Current, which allows you to read or set the motor current for each axis.

Whilst this full UART mod and Linear Advance are done, there is scope for other mods to this board, adding an extra Z motor instead of full UART (or as well as, if you want to solder to the STM32 chip), PWM hotend fan control.

Finished Article

So the mods are done, I guess you want to see a picture of it? See Fig.6 on the next page.

My mod looks slightly different to the method as it's laid out in the document. This is because I started with Wong Sy Ming's method using PA3 (Red wire), then used PA7 (Purple, Z-endstop) for X, PA13 (Yellow, SWDIO) for Y and lastly PA14 (Green, SWCLK) for Z. I used a random piece of stripboard and a female pin socket on the SWDIO/SWCLK port, as it meant less cleanup at the time if things went wrong. On the 4.2.7 board, the header is unpopulated so that will be easier/cleaner to solder to.

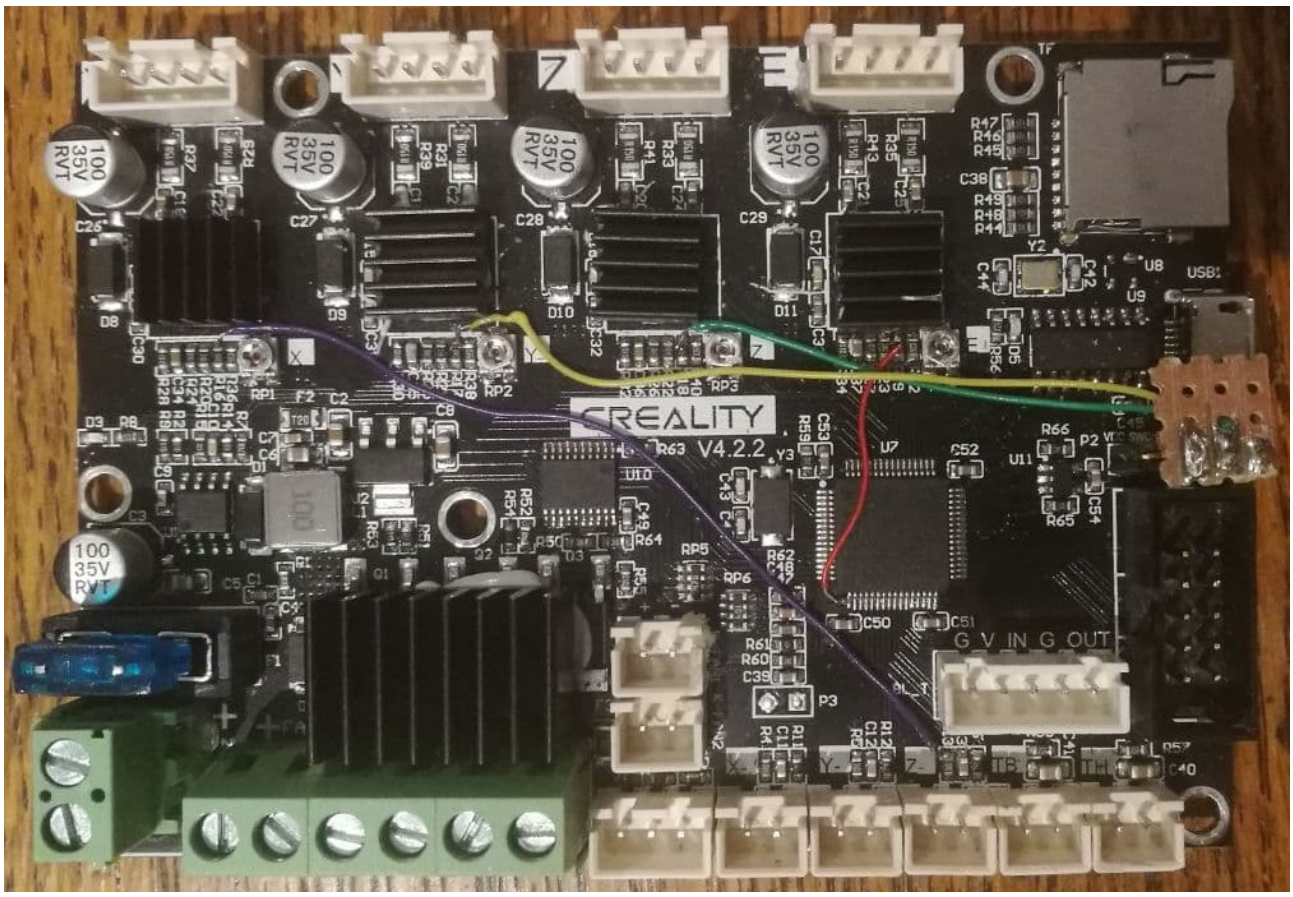


Figure.6

Some notes on how I figured out the mod

It might interest some people just starting out doing hardware mods on how I managed to figure things out. The following is a quick rundown on necessary steps to achieve this:

1. I was originally attracted to this because Wong Sy Ming posted the original mod instructions to Facebook™. As such it was the springboard to allow me to follow on from the good work done by the author.

2. Leg work... With most hardware modifications, it requires some leg work. That means getting data sheets for the STM32F103x and the TMC2208/TMC2225 chips, physically identifying the pin connections with a jewellers loupe and using a digital multimeter in continuity mode to confirm the pins on the STM32 are connected to the respective pins on the headers. Also a DMM in continuity mode to check the state of pins on the TMC2225 without the need to have the board plugged in. For instance, you can check whether the DIAG pin is tied low by putting a probe on one of the TMC2225 GND pins (pins 14/28) and the other probe to the far side of the resistor that the pin is connected to. You can do the same with the MS1 and MS2 pins to see what microsteps the TMC2225 chips are configured for.

You can also check the PDN_UART pin to see whether the TMC2225 reduces hold current or not. (Hint, it does not, so this might be somewhere that you could look if you wanted to increase current whilst reducing the heat on the driver chip).

3. More leg work, the soft kind... As this hardware mod requires firmware changes, it obviously requires you to roll your own firmware, we edit 3 files:
Configuration.h
Configuration_adv.h
pins_CREALITY_V4.h

These are the files we edit for the original Linear Advance mod. Thanks once again Wong Sy Ming. As we're expanding on the original mod, we have to go through these files, check whether any pins we want to use are already in use, undefine them from their original purpose and #define them for our own use, adding our own defines for previously unused pins at the end.

Then we need to edit the right defines in the configuration files to enable linear advance, TMC UART mode, debugging mode for more detailed information from the driver chip and any defines that are necessary to set our driver chips up with default Creality™ based settings.

As it stands, it's not a particularly difficult mod. It just looks like it might be to the untrained eye. The board does have a lot of pins and headers but it's really not much more than a glorified Arduino development board, a handful of MOSFETs and some motor driver chips. The lack of schematics is more of a challenge, but not a show stopper.

If you're new to coding, Marlin looks complicated but that's because it's modular and deals with so many different combinations of hardware and upgrades. Once you have your pins/configuration files edited and have set the correct environment in VSCode, everything else is pretty much done for you. You just have to build it at that point. Figuring it all out just requires focus and plenty of CTRL+F use.