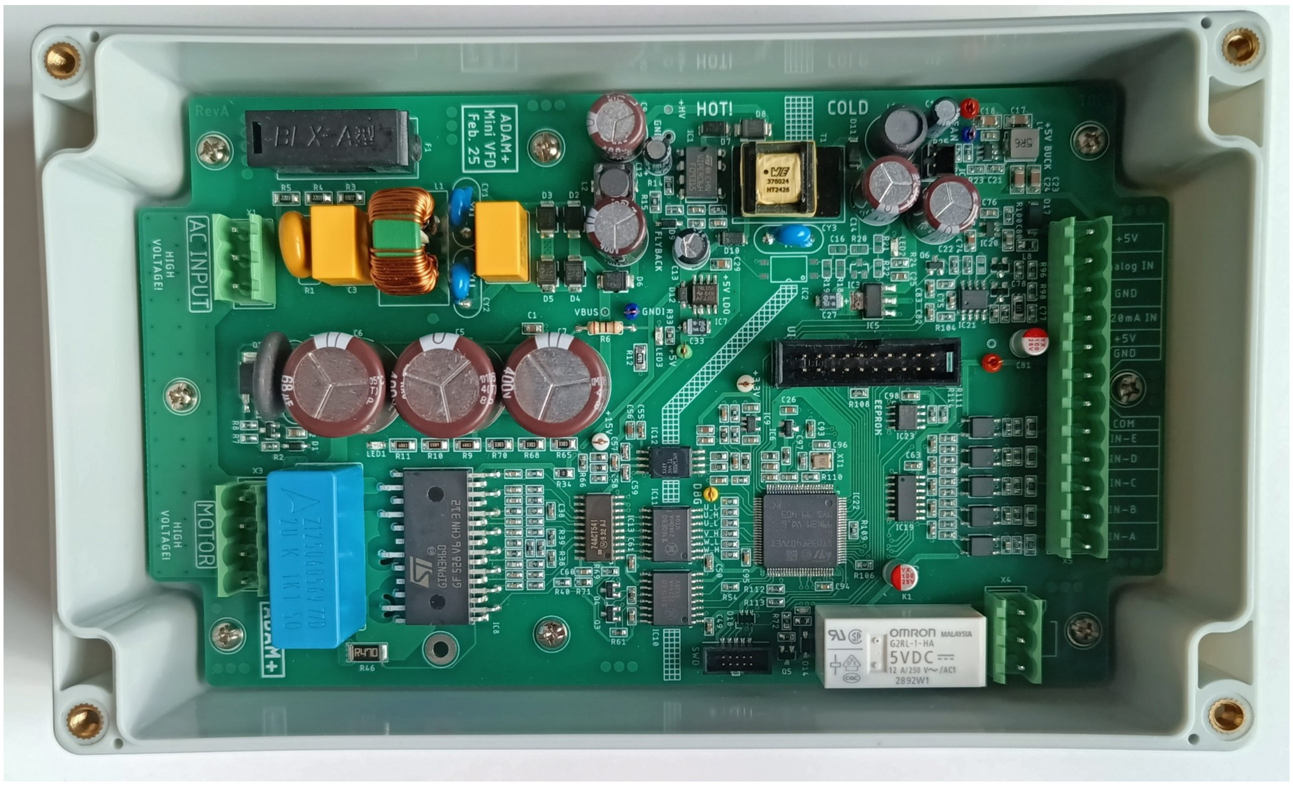
Mini VFD

Michael Adam
michael.adam4@proton.me



General

I've always been fascinated by VFDs (Variable Frequency Drives), but never had the opportunity to develop one professionally, as I don't work in that field. This project represents my personal journey of building a VFD more or less from scratch. Along the way, I've made good use of various helpful application notes provided by IPM (Integrated Power Module) manufacturers, which have greatly supported the development process.

This is a completely independent project — there's no sponsorship or external funding involved. All components were purchased from LCSC, DigiKey, and AliExpress.

You're very welcome to use, study, modify, and build upon the information, schematics, code, and ideas shared in this project. If you publish your own work based on it, I'd appreciate it if you mention me as the original source. That said, everything here is shared as-is, and free to use – in the spirit of learning, transparency, and open hardware.

Important

⚠ Safety Notice

Please note that this project involves **high voltages** (up to and beyond 325V DC), **high currents**, and components that can pose serious hazards if mishandled. This is **not** a beginner-friendly project. Anyone attempting to replicate or modify it must have a strong understanding of:

- Electrical safety and protective measures
- Power electronics
- Isolation techniques
- Thermal management and safe power supply design

Always work with **proper protective equipment**, use an **isolation transformer**, and perform tests with care. You are fully responsible for your own safety.

⚠ Disclaimer

This project is provided for educational and experimental purposes only. I am not liable for:

- Any damage to equipment, hardware, or software
- Any kind of personal injury, electric shock, fire, or property loss
- Any unintended consequences arising from the use, misuse, or misunderstanding of the content

You use this material **entirely at your own risk**. No guarantee is made that the circuits or designs will work reliably, safely, or under any specific conditions. Especially the software is still work in progress.

⚠ Compliance

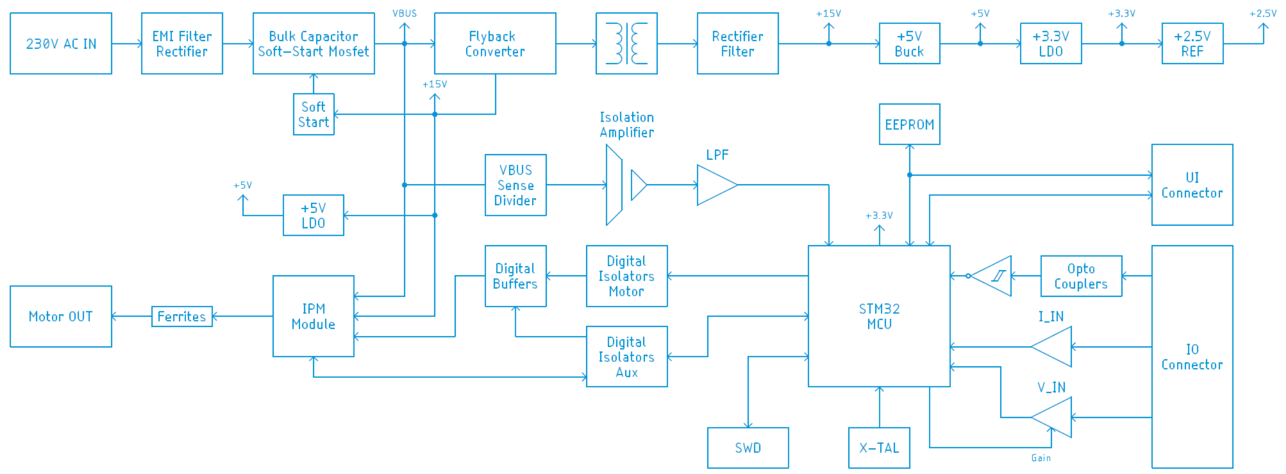
The final device has never seen the inside of an EMC lab! It would most likely fail most of the tests (just like many reference designs from reputable manufacturers, by the way).

Design Concept

Since this is a pure DIY project, there were no strict specifications. But there are still some, to have an idea and plan to stick to it. The main specs are:

- Design for tiny motors around 1A per phase
- 230V AC Input (could also be a universal input)
- LCD Display
- Output Frequenvy up to 150Hz
- Over current protection
- Voltage Sensing
- Fused Aux Output
- Digital/ Analog Inputs
- Easy to solder parts for easy rebuilds (no BGA, QFN etc…)
- Transparent enclosure

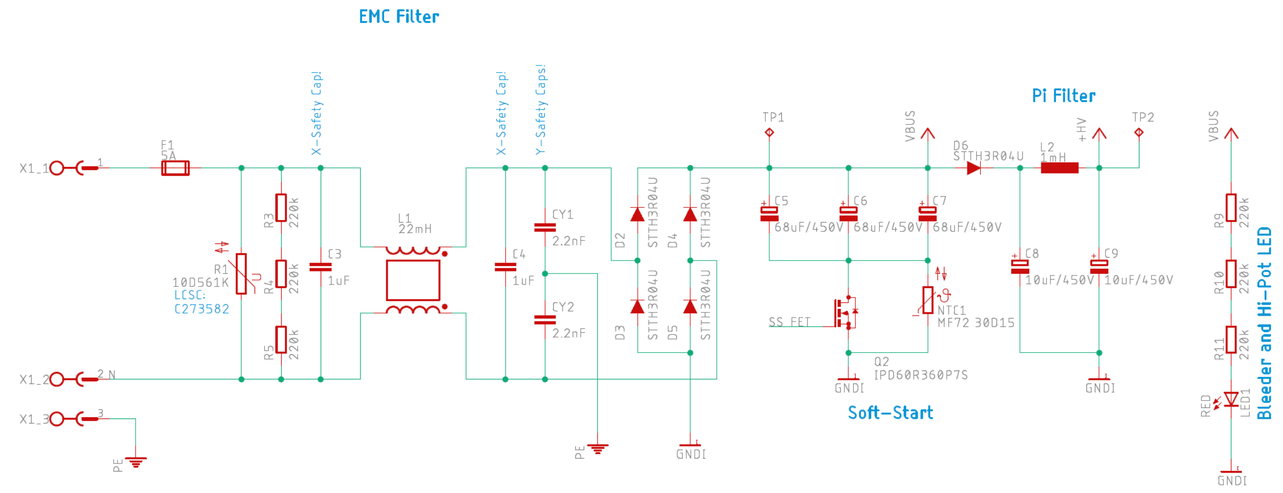
On the picture bellow you can see a very rough concept of the VFD:



It’s pretty straight forward: the mains AC voltage goes through the EMC filter to the rectifier, after rectification to the bulk capacitors, the bulk capacitors are equipped with a soft-start circuit, that bypasses an NTC at the negative pole of the capacitors. The fly-back converter supplies both the secondary and primary side with auxiliary voltages. On the primary side there is a 15V rail, needed for the IPM module, and on the secondary side, a very unprecise voltage, that gets converted into a stable 5V rail. LDO’s are regulating the voltages to logic levels (5V @ primary and 3.3V @ secondary). The IPM has two supplies, one VBUS (325V) supply and an auxiliary 15V supply for the internal gate drivers. Signals to the IPM are coming from the MCU, through some digital buffers for safety, isolated by digital isolators. There are two digital isolators, one for the motor signals and one for other miscellaneous signals. To the able to adjust the motor’s voltage, we need to monitor the VBUS rail, this is realized with a potential divider, an isolating amplifier and some filtering and signal adequating in front of the ADC within our MCU. The brain of our VFD is an ARM Cortex M4 STM32F407 microcontroller, which might be an overkill for the tasks it has to do, but it’s better in my case to oversize something and have enough headroom to expand the functionality. An external I2C EEPROM is being used to store user parameters. Optocouplers are isolating the digital inputs. The analog signals are going through some protection to opamp buffers and then to the MCU’s ADC.

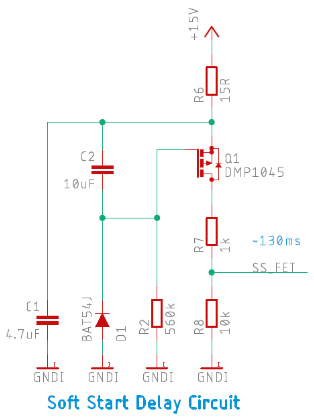
Circuit Description Mainboard

AC Input section:



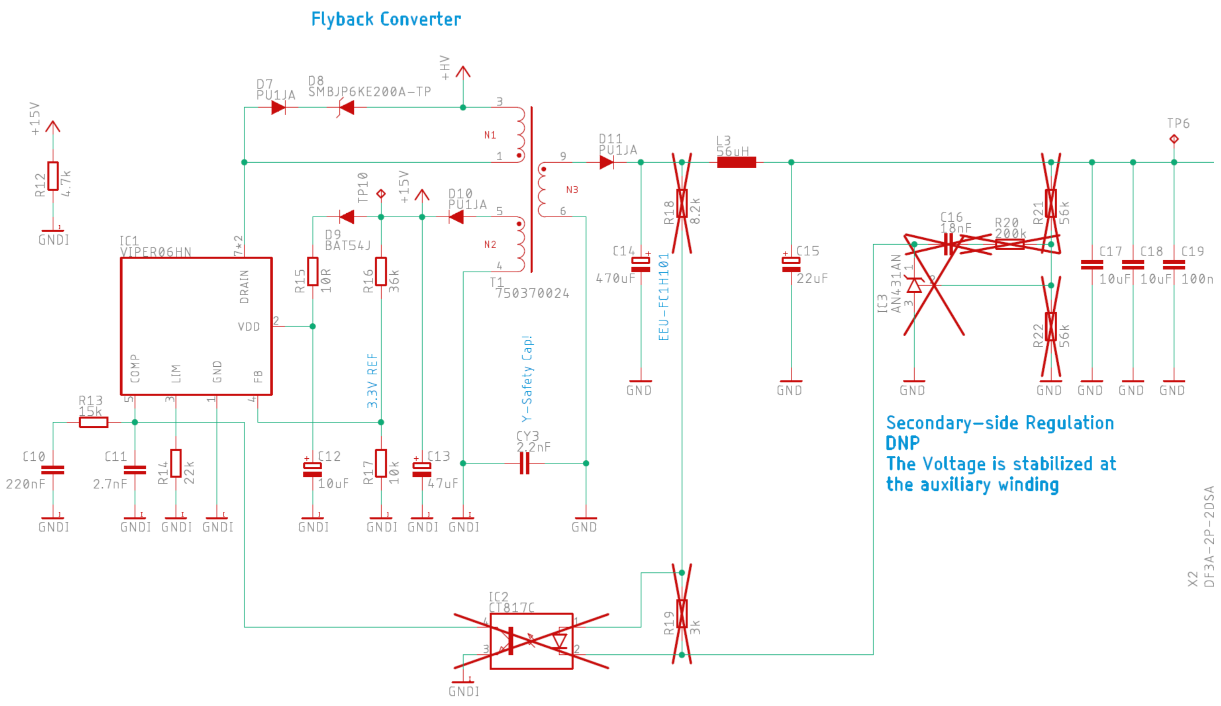
EMC filter on the left, discrete diodes bridge rectifier and some electrolytic s that have a soft-start circuit. When the device is powered the Mosfet is open and the NTC righ resistive, after around 130ms, the Mosfet closes and bypasses the NTC. VBUS goes directly to the IPM. A Pi filter is placed between VBUS and the fly-back circuit, to reduce noise coming from the fly-back converter. An LED shows if there is high potential at the caps and at the same time discharges them.

Soft Start delay circuit:

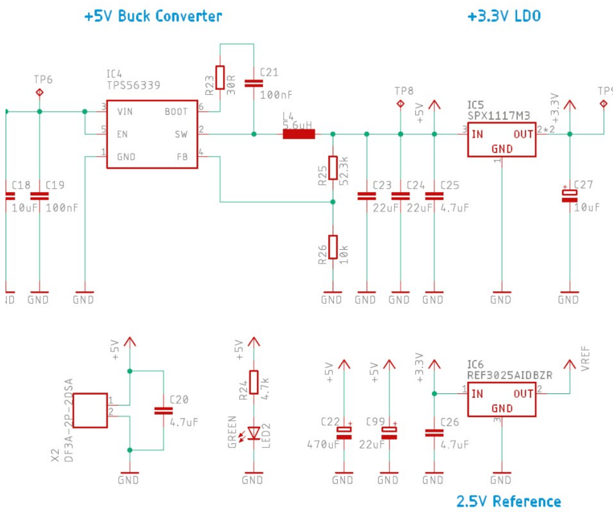


The small circuit above, is a quick and dirty way how to add some delay to bypass the NTC.

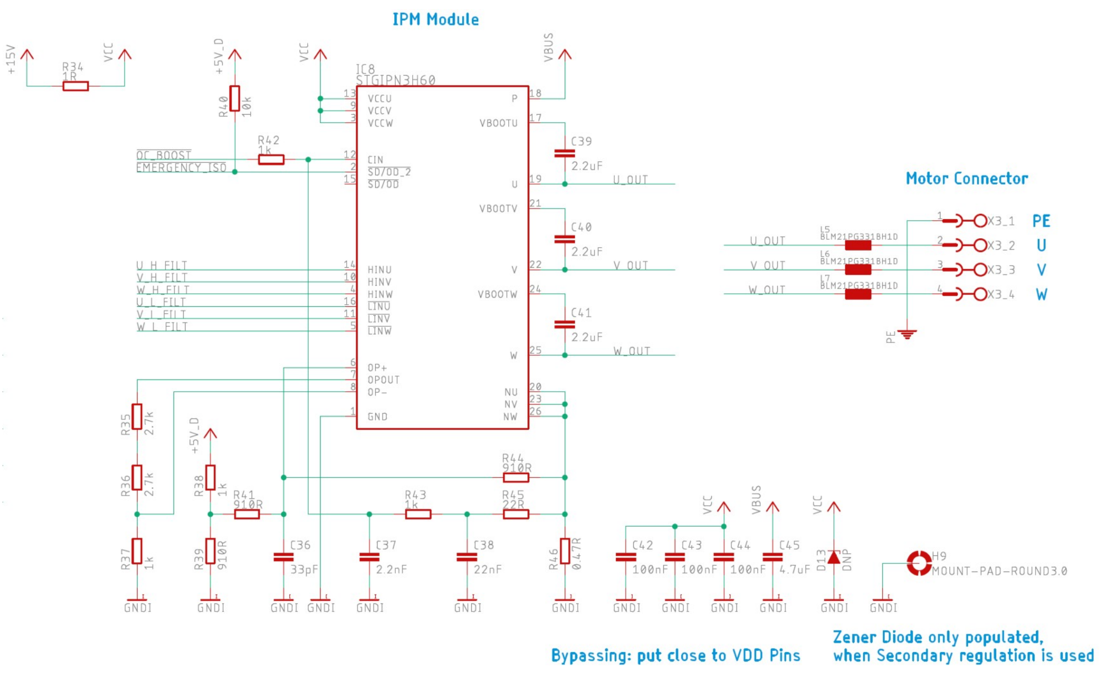
Flyback converter:



The fly-back converter supplies the primary side with a clean and regulated 15V and the secondary with something around 12V, the primary side regulation is not really working that great, but we can catch it in the buck converter. Secondary side regulation is placed on the board, but not populated, I am more interested in a clean auxiliary voltage rail on the primary.



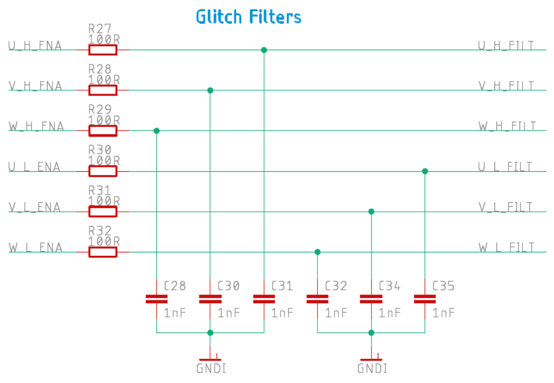
Simple synchronous buck converter, all values were calculated by Texas Instruments’ WEBENCH. Small LDO for the 3.3V rail and a reference that goes to the ADC’s reference pin of the MCU.

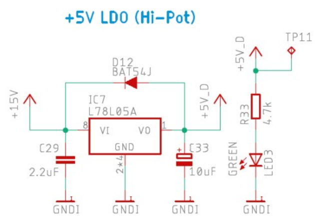


IPM Modules are just perfect for these kind of projects. No Gates to drive, no slopes to control, everything is done internally, we just input logic values and that's more or less it. It is crucial to bypass the IPM in the correct way, as described in the data sheet and application notes. C42-C44 are bypassing the 15V Rail and a big 4.7uF foil capacitor C45 does the same for VBUS. Before going out to the motor connectors, there are some ferrite beads to filter out at least some of the high frequency content, coming out from the IPM. Be aware U_OUT, V_OUT and W_OUT are going with very high dv/dt from 0V to 325V! Remember, in the «real world», the motor cables must be as short as possible and the VFD as close as possible to the motor! The best case if the VFD sits directly on top of the motor, so all the high frequency noise stays inside the metal housing of VFD and motor.

The used IPM is more made for something like HVAC and other machines, where the VFD is integrated in the device, and less for a standalone VFD. But since this project is more a demonstrator than an actual product, it's fine. Therefore it has some built in over current protection, we just have to add some passives externally. This part of the circuit has not yet been fully tested and tuned! The values have been taken from data sheets and application notes.

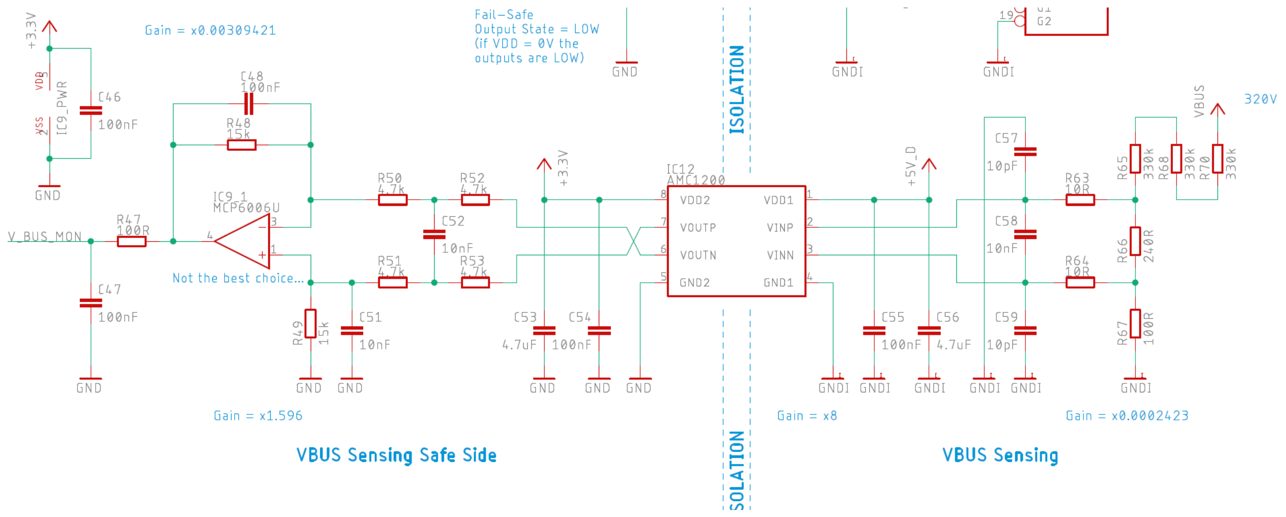
Some auxiliary circuits on the hot side:





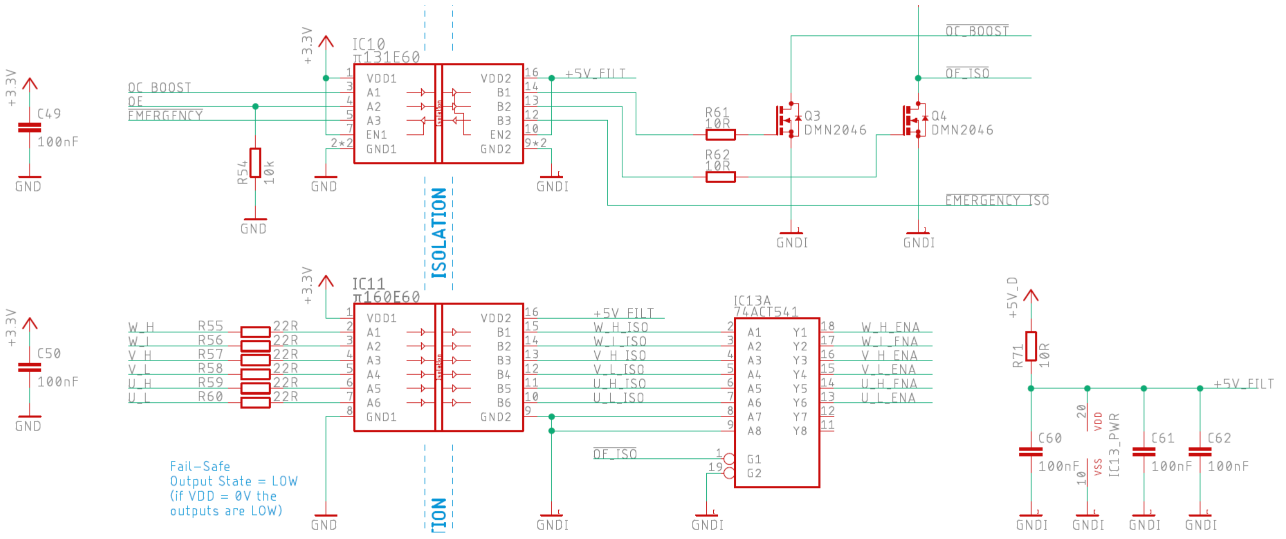
Nothing special here, just some low pass filtering, to avoid interference's on the six control lines. Those lines are critical! A false signal state could damage the IPM or at least disturb the function.

VBUS Sensing:



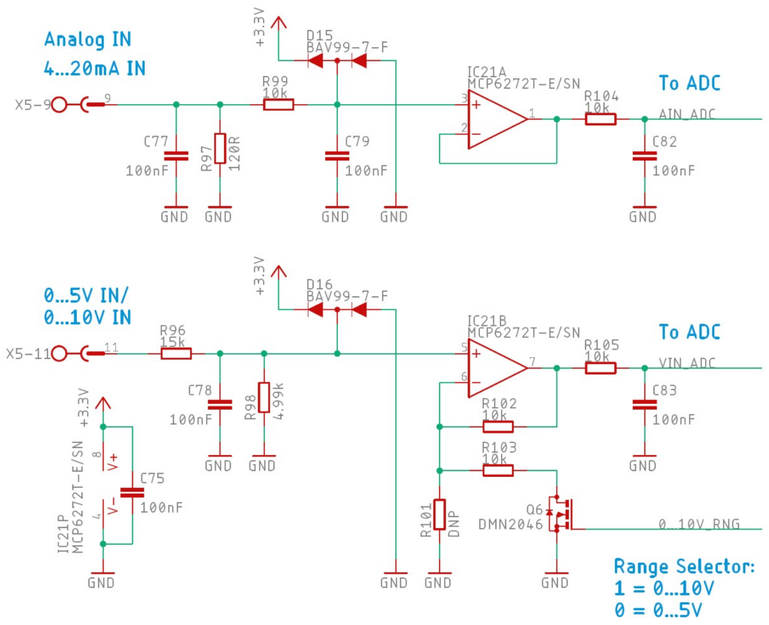
VBUS, which is around 325V DC @230V AC input, has to be attenuated first by a gain of x0.0002423, before going into the isolation amplifier, which has a gain of x8. The final stage before the ADC has a gain of other x1.596, so the resulting gain is x0.00309241. @325V DC, we get around 1V at the output, going to the ADC, which is not that great, since the reference is 2.5V, so there is some of the ADC range lost, but it should be enough. The passives are filtering unwanted frequencies and transients. The opamp is not the best choice, I just had it lying around.

Digital Isolators:



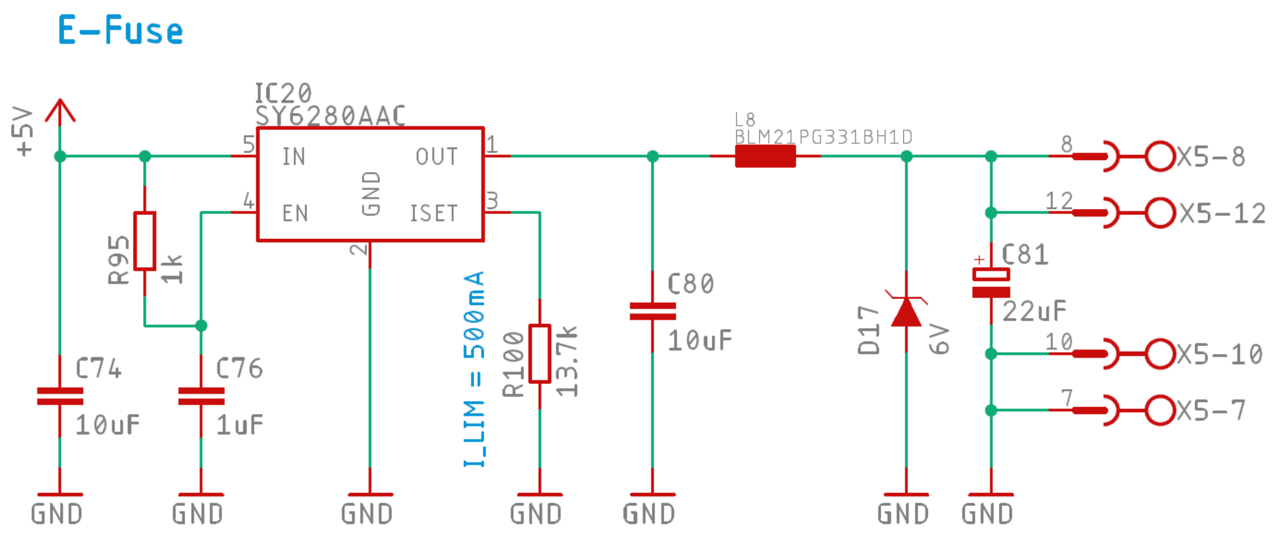
These digital isolators are safety critical parts, they isolate secondary from primary. 74ACT541 is a digital buffer, which is being enabled when the VFD is enabled.

Analog Inputs:



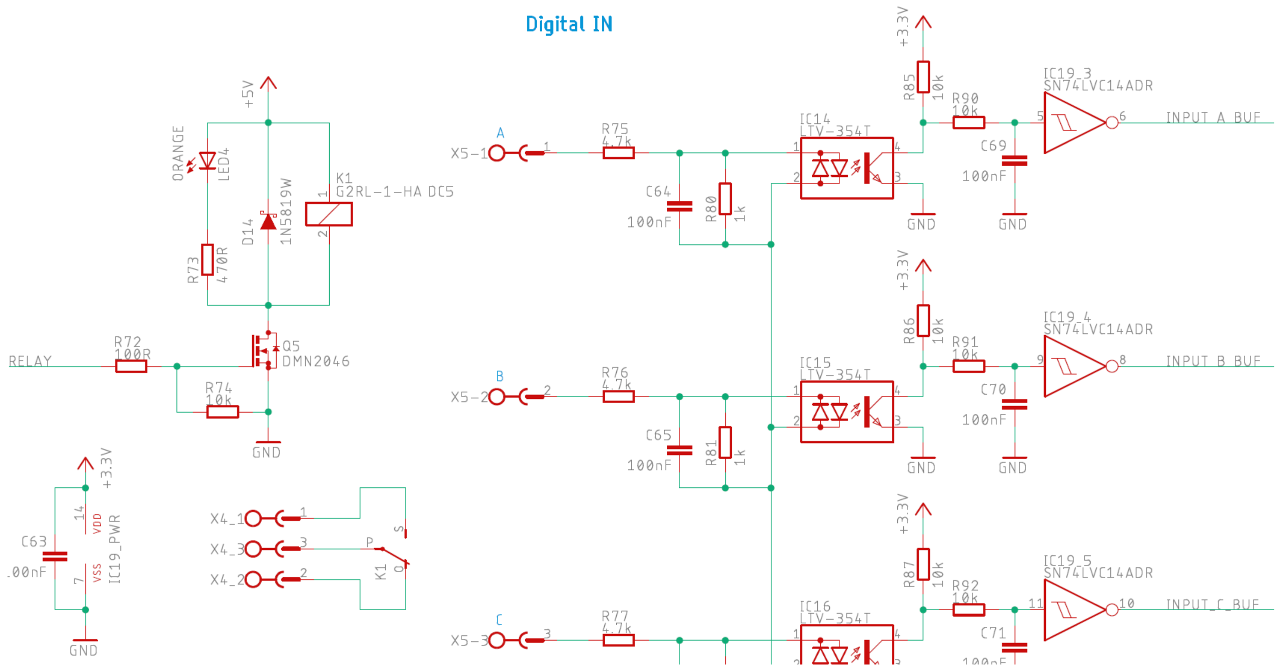
R97 acts as a I-V converter, the resulting voltage is getting filtered, buffered and goes through an other filter to the ADC. With the voltage input, it is a bit more complicated. VFD’s usually have 0...10V analog inputs, which is an industry standard. They also provide 10V, which the user can also connect to a potentiometer and analog input, to control the motor remotely. On the secondary side, the highest voltage is 5V. To not loose ADC precision, there is a range selector, which can change the gain from x1 to x2 of the opamp. So at 0...5V and 0...10V input, the output will be 0...2.5V.

E-Fuse:



To protect the internal circuit, there is an E-Fuse, limited to 500mA. A TVS diode protects from ESD pulses.

Digital IO’s:



There is only one by a relay galvanically isolated output. This output can be configured via the GUI, what it should do. Same for the digital inputs. They are all isolated, but not between each other!

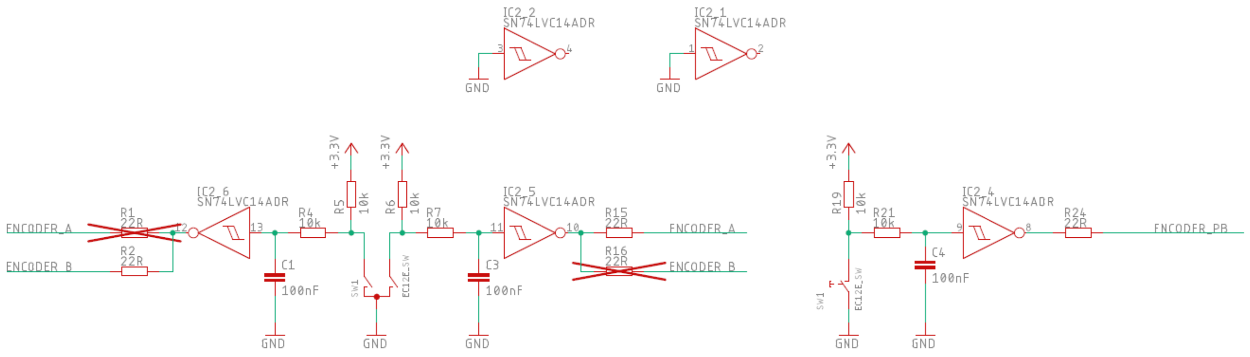
Last Page:

Is self explanatory, just an MCU, with an x-tal oscillator, a connector to the user interface PCB and an EEPROM for user parameter storage.

Circuit Description UI Board

The UI (User Interface) board is mounted at the top cover plate, giving access to a display, 4 illuminated buttons and a rotary encoder. Let’s go through the simple schematic:

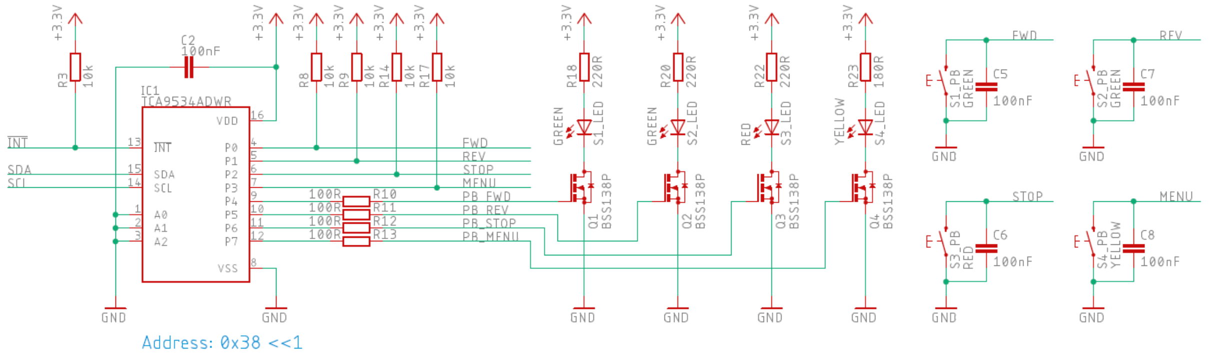
Debouncing:



The debouncing I took very serious for the rotary encoder, I already had some painful experience in the past with them. The 20 cents are well invested in my case. For sure it would be possible to solve it completely in software, but this would have taken a lot of time. Like now, the aggressive filter is getting rid of all the bounces and the schmitt-triggered inverter is converting the banana-like signal into a nice LVCMOS signal. In case the encoder-mode counter in STM32 is counting in the wrong direction, I added an option to change polarity of the encoder, which is much easier than messing around with the code or settings.

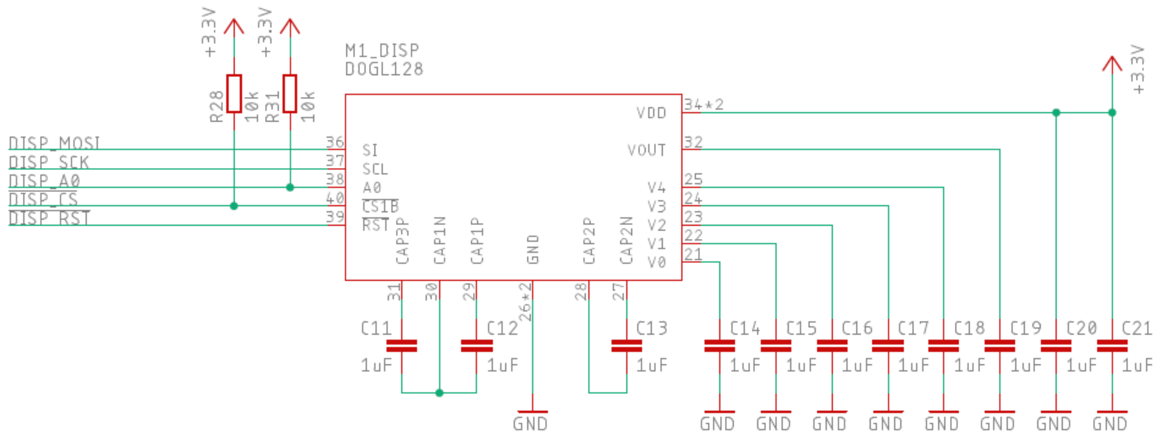
Debouncing for the encoder push-button is not necessary, but since there are some inverters free, why not.

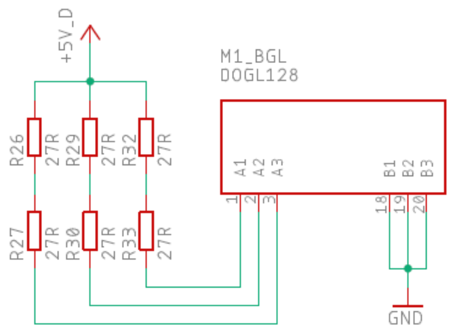
IO Expander:



Through this IO expander, the MCU can **poll** push-button states and switch the LEDs. The communication protocol to the expander is I2C, which is not ideal, since it could make problems in harsh environments (noise/ transients..), but it this project it is okay.

Display:





I use a DOGL128 LCD clone from Aliexpress. They work so far as good as the originals. Just the background light is original, ordered from Digikey. The datasheet precisely describes, how to connect and drive the LCD. To avoid glitches during MCU reprogramming, I added pull-ups to the A0 and #CS lines.