WHSPLC

ESP32

KNX via neuem ST-IC

PoE

Ethernet

CAN

One-Wire per i2C-2482

RS485-Modbus

SPI für lokalen Bus

MicroUSB-Connector für die 4 JTAG-Signale und Masse

Weiterer MicroUSB für serielle Schnittstelle

Alternativ: USBC für alles!

Senkrechte Platine

Ladeschaltung für SUPER-CAP

Statusanzeige mit i2C-Display 0.91 mini oled

Stromversorgung 24V

Lokaler Bus per IDC und per RJ45

24V-Versorgung wird ständig überwacht. Wenn diese wegbricht, dann ist ein Kondensator mit 1000uF noch in der Lage

# Hardware

Einbau in ein 1TE schmales Hutschienengehäuse

Basis mit einem ESP32 Dual Core, und RTOS. Ein Core ausschließlich für SPS, der andere für WEBUI. 16MB Speicherplatz. Für selbstlernende Geschichten muss es noch einen hochperformanten Chip zum reinstecken geben. (NanoPi Neo Core2)

Ethernet-Schnittstelle, Can-Schnittstelle, I2C-Schittstelle, RS485-Schnittstelle, 1Wire-Schnittstelle, LIN-Schnittstelle

Eine SPI-Schnittstelle exklusiv für die Modularität. Es soll vorgesehen werden, dass sich Module direkt

Für den SPI-Prototyp verwenden wir (endlich) die RJ45-Stecker für In und Out getrennt. Auf den Bus werden 5V Betriebsspannung gegeben und mit einfachen Längsreglern auf den Modulen auf 3,3V gebracht. Auf den einzelnen Modulen arbeitet ein STM32F030 für 8pol ios oder ein STM32F103 für 16pol.

Intern i2c-Bus mit BMP280, Grafisches Display, Betriebsartenschalter, DrehDrücksteller

# Lokaler Bus

Im Folgenden wird die Kommunikation des CPU-Moduls mit den angeschlossenen IO-Modulen beschrieben. Das Konzept basiert auf einem großen Schieberegister. Jedes IO-Modul verlängert das Schieberegister.

Physisch besteht der Bus aus Versorgungsspannung 24V, Versorgungsspannung 5V (lokale Linearregler für 3V3), Masse, einer Taktleitung, einer kombinierten Strobe und Reset-Leitung (Kurzer Puls=Strobe, langer Puls=Reset; ggf noch eine OneWireProtokoll implementieren. Außerdem eine Ringleitung für die Daten. Hinsichtlich der Ringleitung gilt. Der Hinweg wird in jedem Modul durch das Schieberegister unterbrochen, der Rückweg ist durchverbunden. Jeder Datenausgang im Hinweg ist durch einen passiven Pullup auf 3V3 gezogen.

Um Betriebsbereitschaft anzuzeigen, leiten die Module ein an ihrem Data-In anliegendes LOW-Signal an Ihren Data-Out-Ausgang weiter. Wenn dieses LOW also wieder am Master angekommen ist, dann weiß dieser, dass alle Bausteine betriebsbereit sind und die Ringverbindung geschlossen ist.

Dann sendet der Master ein Reset (min 100ms auf Low ziehen). Nach dem Reset sendet jedes Modul seinen 32bit-TypID (globale Registry) weiter. Der Master hat die TypID 0. Wenn diese 0 also wieder am Master ankommt, wurden zuvor alle TypIds der Module empfangen. Der Master kann nun ein interner Abbild der Module erstellen. In Abhängigkeit des Moduls werden nun 1…N Initialisierungsworte gesendet. „0“ wird dabei als NOP interpretiert, FFFF als „Ende Initialisierung).

Ab dem Ende der Initialisierung arbeiten die Module entsprechend der Initialisierung. In jedem Zyklus empfangen sie ein Wort und senden ein Wort weiter. Reine I-Module interpretieren die empfangenen Daten stets als Befehl (z.B. Bereichsumschaltung). Reine O-Module können entweder nicht rekonfiguriert werden oder interpretieren das MSB der empfangenen Wortes als Zeichen für „Daten“ oder „Befehl“.

In den Folgenden Zyklen schreibt dann der Master passend für die Leseschieberegister aller Bausteine raus. Dann Setzt er Strobe. Die Bausteine übernehmen ihr Schieberegister und stellen auf Schreibschieberegister um. Der Master nimmt strobe zurück.Mit den dann folgenden Taktimpulse senden die Bausteine dann also ihre Daten raus in Richtung Master. Sobald sie ihre Daten losgeworden sind, stellen sie wieder um auf Leseschieberegister und der Zyklus beginnt von neuem.

Problem: Insgesamt könnte mehr geschrieben als gelesen werden und umgekehrt. Machts nichts, oder???