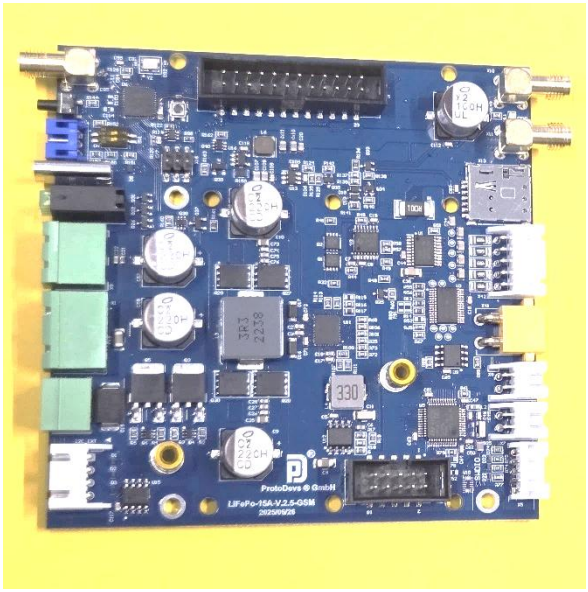
Industrial-grade LiFePo4 v.2.5 Charger module

With ESP32 / WiFi / Bluetooth / GSM / GPS features

by ProtoDevs® GmbH

Datasheet (Rev. 18.07.2025)

PD.Charger_GSM Family description



The PD.Charger_GSM is a smart charger controller module which manages the **LiIon** / **LiFePo4** battery charging with extended monitoring functions and **I2C** / **UART** to control and monitor **MPPT**-enabled Charger chip, Gauge chip, BMS chip, voltage/current sensors and onboard Microcontroller (**ESP32 + 16 MB Flash** memory).

PDCharger-GSM v.2.5 module has the Input/Output **Voltages**, **Currents** and **State of charge**, each **Cell voltage** monitoring together with **connectivity**: **WiFi** and **Bluetooth** implemented by onboard **ESP32** microcontroller (Arduino-compatible) and **GSM/GPS** from SimCom **SIM7080-G** module. All the **RF interfaces** have SMA connectors for easy connection of the corresponding antennas.

The device charges a **LiFePo / LiIon** battery from a wide range of the input sources including external **DC adapters** and/or **Solar Panels** and traditional barrel adapter with the precise **MPPT** (Maximum Power Point Tracking) algorithms implemented. The charger automatically sets output voltage by using converter in buck, boost or buck-boost configurations based on input voltage and battery voltage without the host control. The **Dual input source** connectors manage the power flowing from two different input sources.

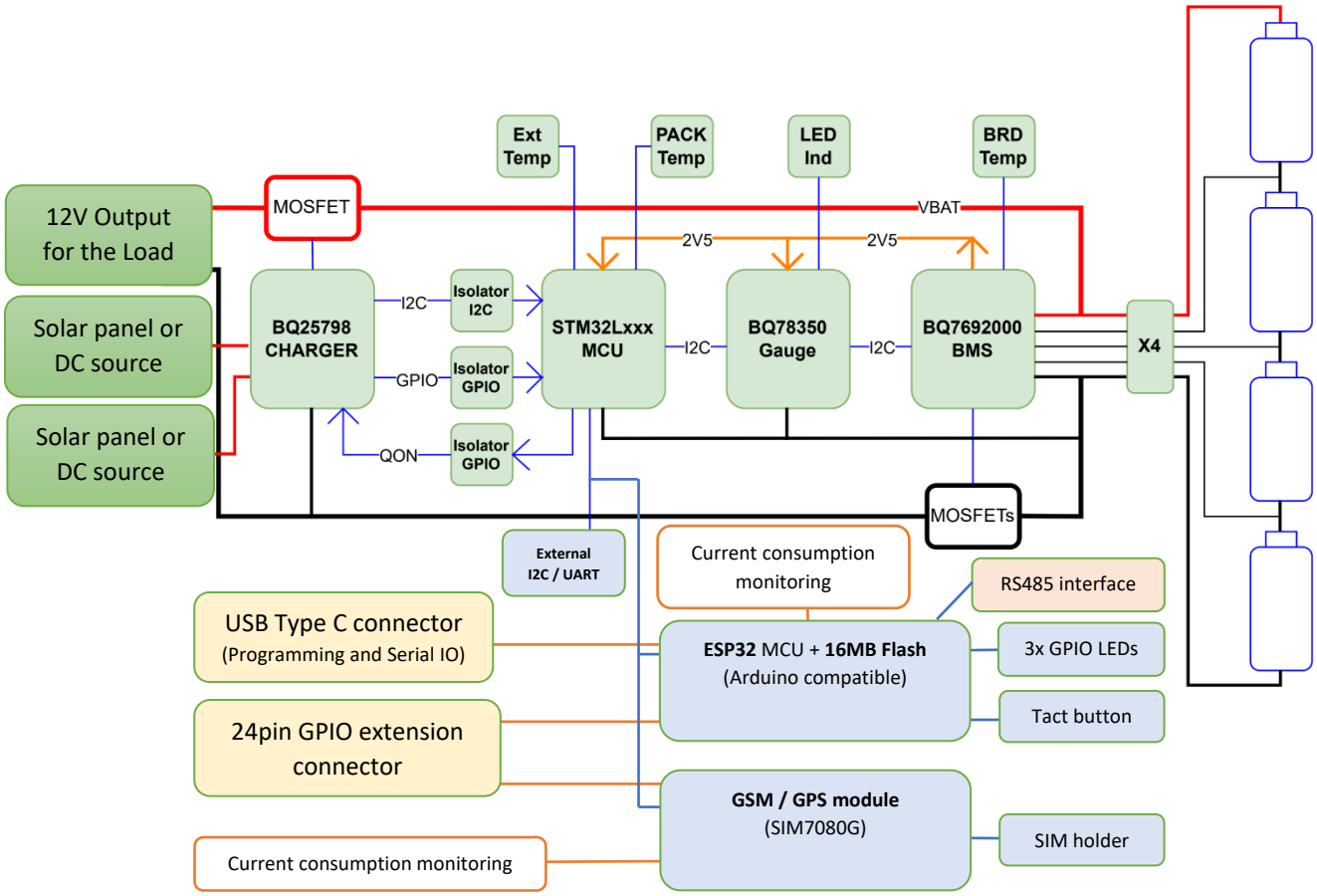
PD.Charger_GSM has a **dedicated STM32** microcontroller which provides **API** to any **I2C/UART** control unit (**onboard ESP32** MCU and/or **external MCU/SoC**) utilizing the power and cost-effectiveness of the latest industry standards. PD.Charger_GSM could be customized for any type of battery and any type of connectivity. Please contact info@protodevs.de in case of HW / FW customization need.

PDCharger v.2.5	up to 12V @ **14A+** load, all the v.2.3 features + MCU ESP32 (Arduino compatible) + GPS/GSM module (SIM7080G)

Table of Contents

Simplified block diagram

(For PDCharger v.2.5)

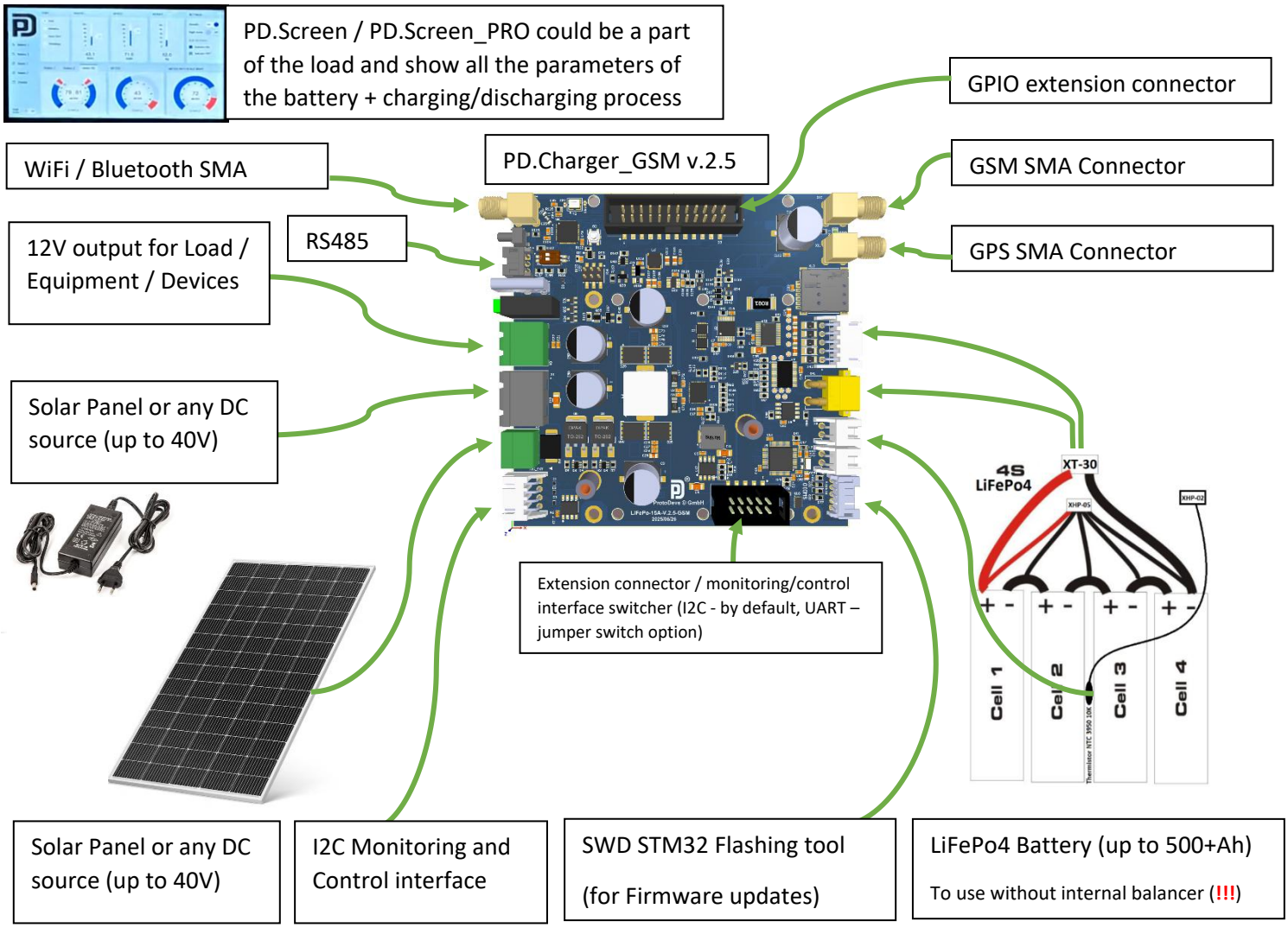


Operational conditions

Operational temperature:	-40 … 85 C
Storage temperature:	-55 to 150 C
Input voltages range (X1, X2 connectors):	3.6V to 40V wide input voltage
ESD rating:	Charged device model (CDM), per JEDEC specification JESD22-C101, all V pins: +-250V
Max batteries charging current supported:	12A max, 5A by default (see API)
Mass:	82 grams for v.2.5
Communication speed (I2C bus):	up to 1 Megabit per second (1 Mbps)

General specifications

PDCharger v.2.5 typical connection diagram:



PDCharger-GSM Controller consists of 6 major blocks:

SMT32 MCU , BMS chip , Gauge chip , Charger chip , ESP32 MCU , GPS/GSM modem module.

The default configuration of the PDCharger v.2.5 (Charger/Controller part) is designed to use all the possible information to provide detailed information regarding the battery status and load status during the idle load operation, charging or discharging states.

The strict requirement is to use LiFePo4 batteries **without any** additional electronics inside the battery pack (load balancers, protection circuits) because everything already implemented on the PDCharger PCB side. In case of additional balancer exists in the battery – please turn off the PD.Charger's embedded balancer with API command and do not expect precise monitoring data.

If there is an emergency case that no other variant exist and the system should work with the battery pack balancer then there is a strict rule to turn OFF the PDCharger internal balancer to avoid two balancers working at the same time. The API command could be found in this document to turn OFF the PDCharger balancer.

Electrical specifications

PDCharger v.2.5 Electrical characteristics:

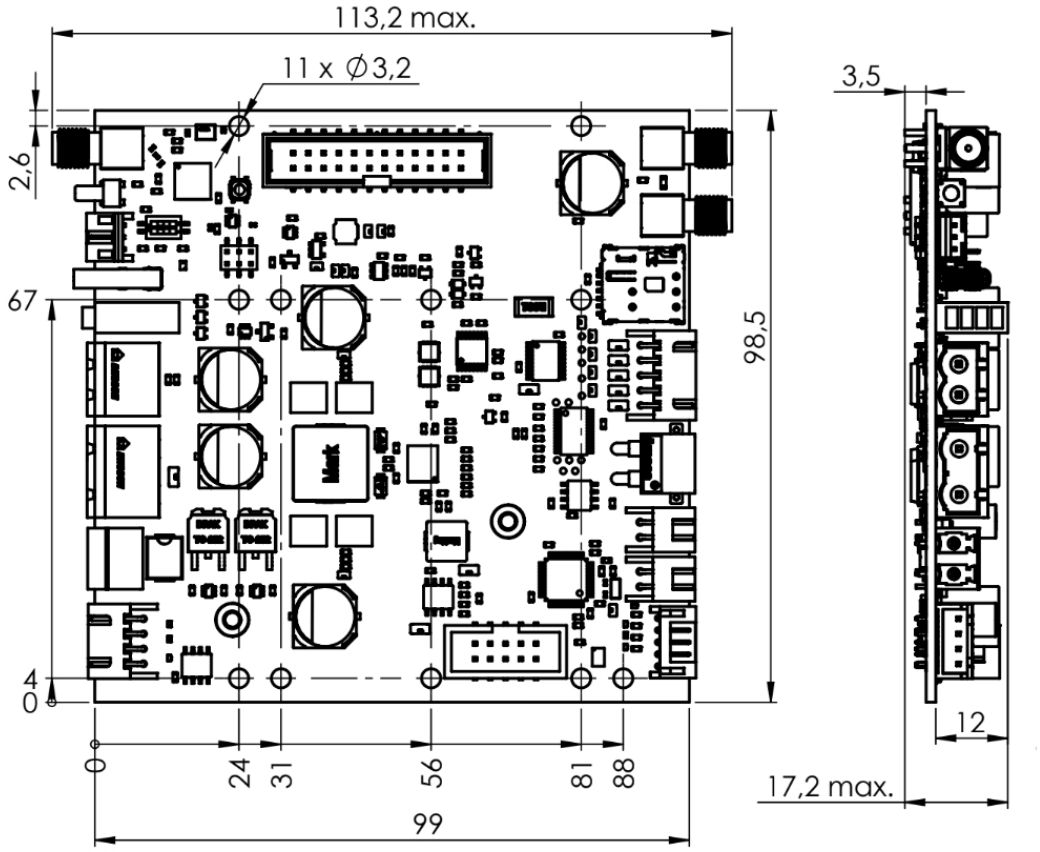
Parameter	Value	Comment
Input Voltage (X1 connector)	40V max	18V recommended
Input Current (X1 connector)	7A max	
Input Voltage (X2 connector)	40V max	18V recommended
Input Current (X2 connector)	8A max	
Total input current (X1+X2)	14A max	
Charger output current (X41)	12A max	7A recommended
Load output voltage (X9)	12V	
Load output current (X9)	14A max	8A recommended
Idle mode current (no load, no charging)	1 mA	Only Charger part measured, ESP32 block disconnected

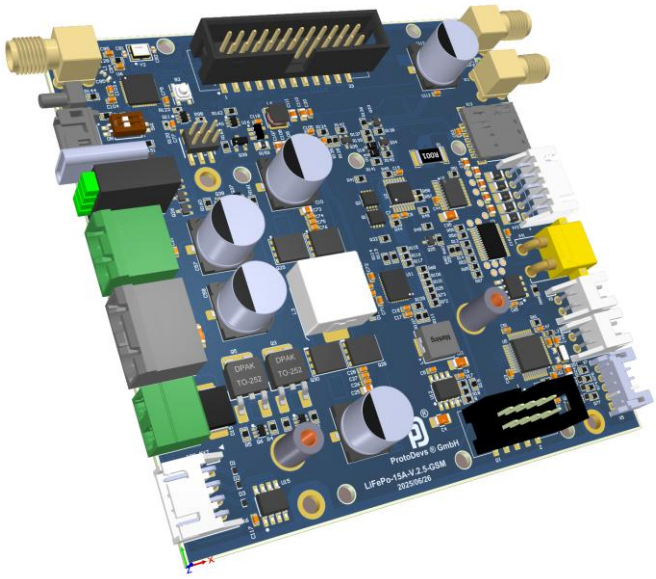
PDCharger v.2.5 default settings (Firmware defined by default):

In progress

Mechanical dimensions (PD.Charger_GSM module)

(For PD.Charger_GSM v.2.5)

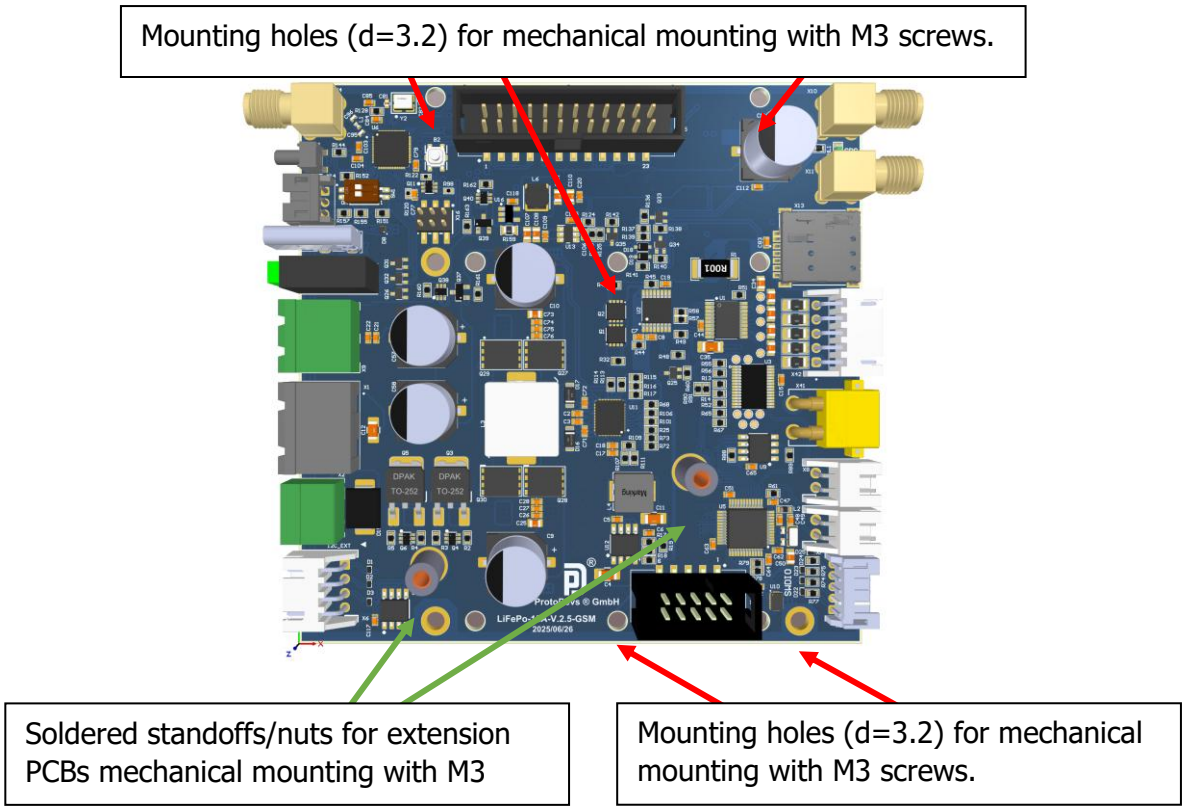




PD.Charger module installation and operating requirements

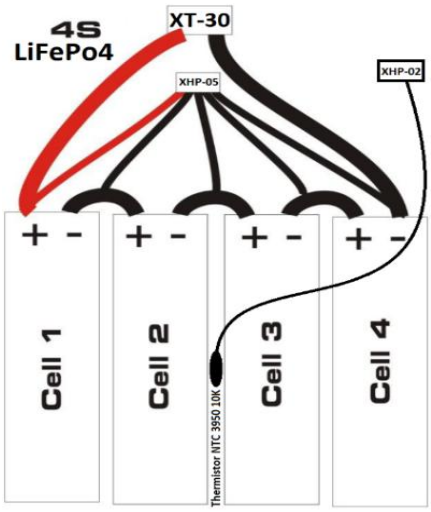
Mounting requirements:

There are 11 mounting holes for the mechanical fixing the charger module inside the enclosure, they marked with the red arrows on the picture:



LiFePo4 Battery balancer connector pinout (a must)

There is a strict requirement for the balancer connector pinout from the external LiFePo4 battery, in other case the Charger module will turn ON the protection and will not work (no output):

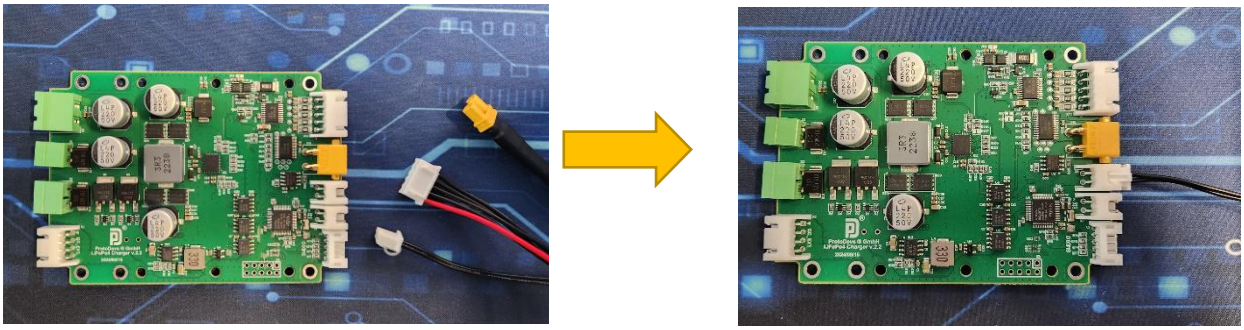


Battery balancer pinout	
XHP-05 connector pin	**connect to**
1 (Red wire)	+ 12.8V
2 (Black wire)	+ 9.6V
3 (Black wire)	+ 6.4V
4 (Black wire)	+ 3.2V
5 (Black wire)	GND

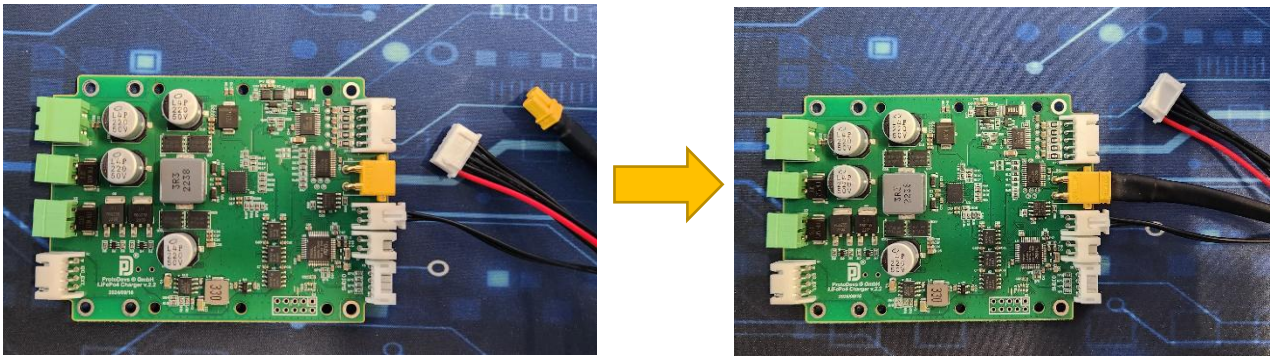
Connection sequence requirements:

There is a special connection sequence exists in order to start the device properly due to the POST (Power On Self-Test) procedure for **PD.Charger** and **PD.Charger_GSM** (PD.Charger v.2.3 was used for the visual instruction, the same sequence must be applied for PD.Charger_GSM):

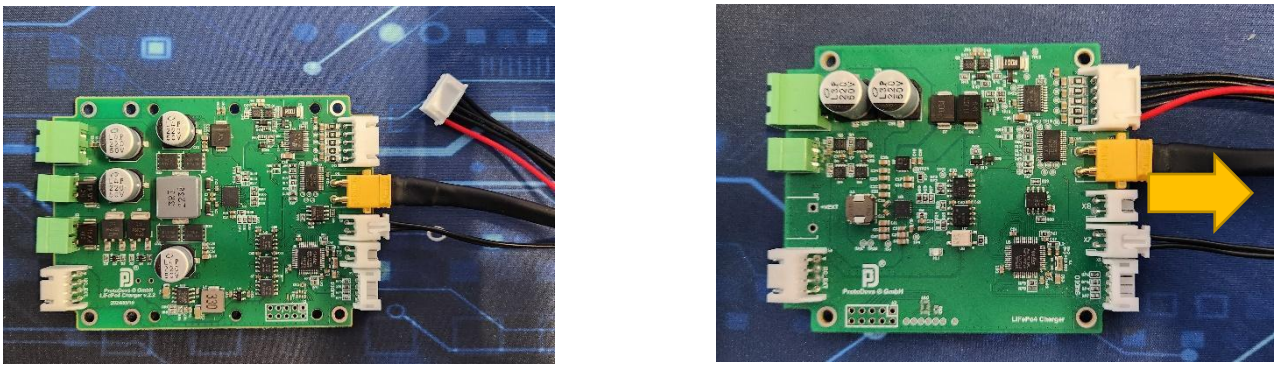
Step 1: Connect Battery temperature connector (2 pin, white -> to X8)



Step 2: Connect LiFePo4 Battery connector (2 pin, orange -> to X41)



Step 3: Connect Battery balancer connector (5 pin, white -> to X42)



Connect the Load **ONLY** after connecting the battery (these 3 steps described)

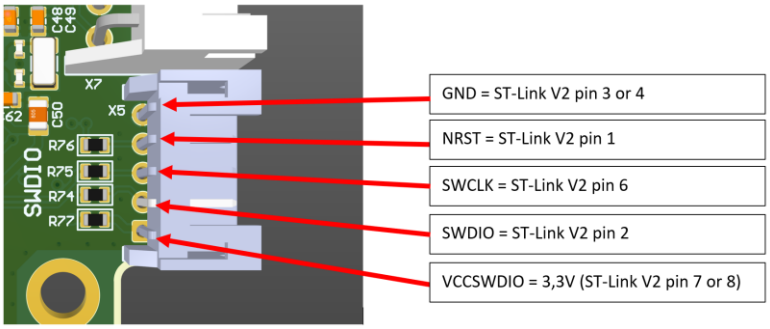
To obtain correct (precise) "state of charge" parameters there is a must to connect 100% charged battery from the very first installation, in other case several discharge-charge cycles are needed for system to calibrate itself.

PDCharger-GSM module programming/flashing recommendations

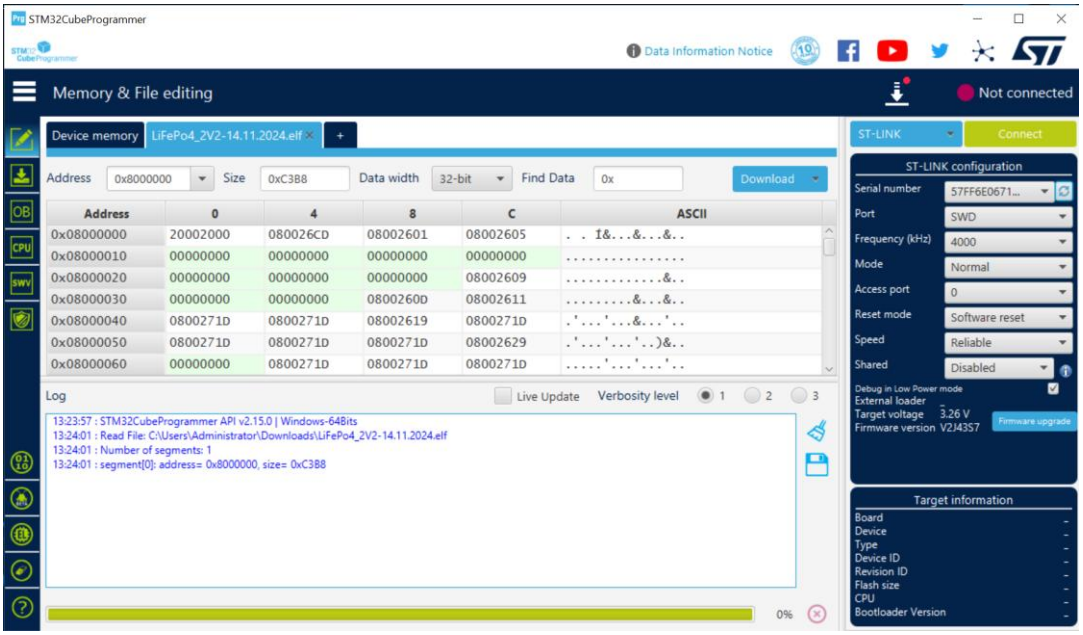
PDCharger module has so-called SWDIO interface to program/flash the Firmware.

If there is a need to update the Firmware then it is recommended to use ST-Link V2 programmer/flasher as the easiest way to proceed.

Connection diagram for programming/flashing the Firmware:



The Software to program/flash the PDCharger module: **STM32CubeProgrammer** (Link)



PDCharger module mounting recommendations

Outdoor installation of the PDCharger modules require certain type of protection and climatic considerations to follow.

The module should be mounted into enclosure which is a solid metal, outdoor plastic or similar material which gives fully weatherproof barrier but also through climatic controls ensures the interior of the enclosure is the optimum environment for running the charger module – no matter what the ambient conditions or temperatures are like.

1. **Install lightning protection devices**

In outdoor environments, lightning protection devices must be installed for the charger placement enclosure and the building where it is located. The main body and casing of the module should be well grounded, and the grounding resistance should be less than **3 Ohms** to ensure that the large current caused by lightning can be discharged in time. Effective lightning protection measures can prevent strong electric and magnetic attacks on the charger module caused by lightning, thereby protecting the safety of the equipment.

2. **Waterproof measures**

Outdoor charging modules must have excellent waterproof performance. Since the outdoor environment is often exposed to the sun, rain, wind and dust, the charger may face serious challenges in waterproofing and moisture resistance. Once electronic equipment is wet or damp, it may cause a short circuit or even a fire, causing serious losses. Therefore, waterproof enclosure design is the key to ensure the long-term stable operation of the charger

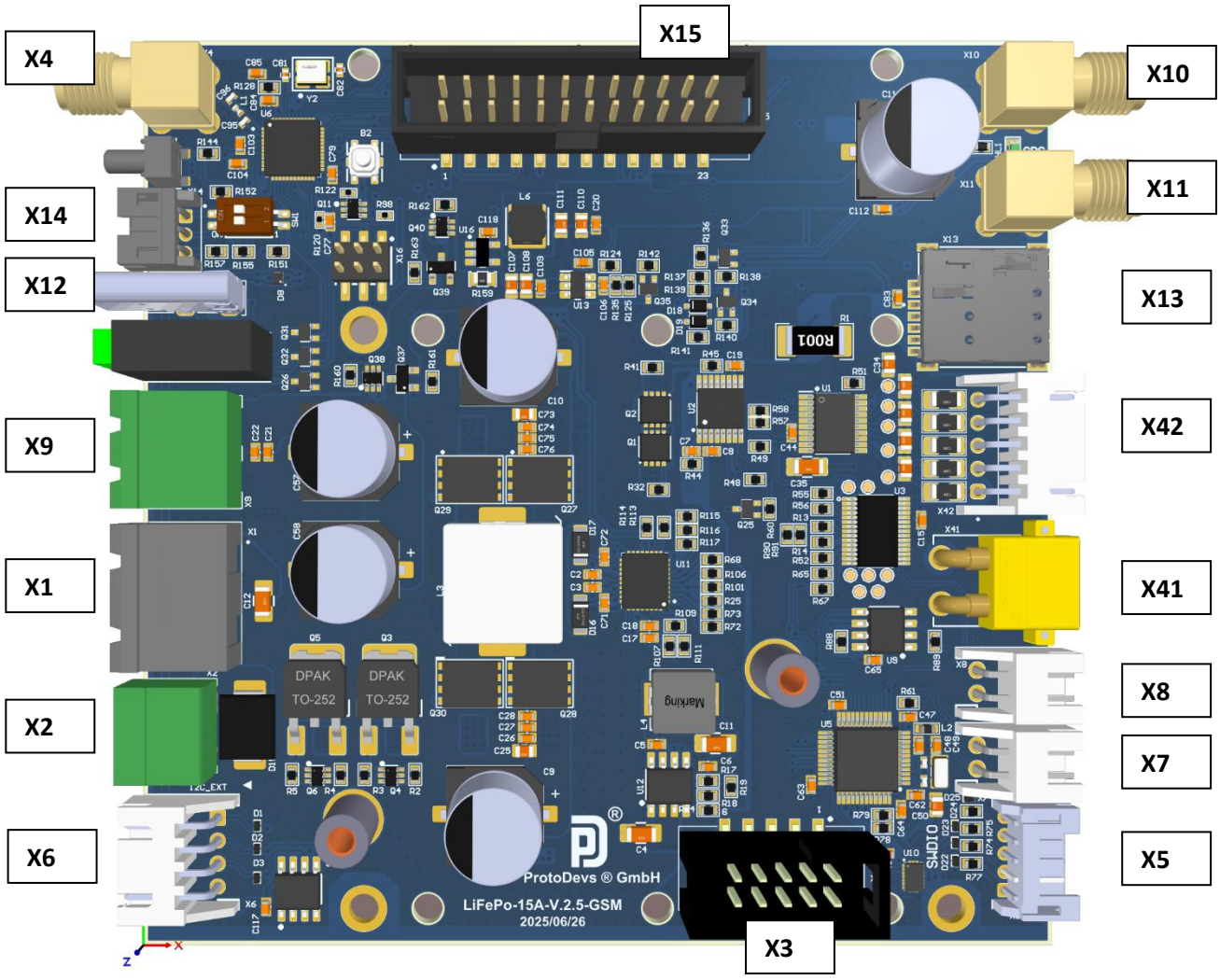
3. **Ventilation and cooling**

In order to prevent the charger module from malfunctioning due to overheating, ventilation equipment must be installed for cooling. Usually, an axial fan is installed above the back of the r rack where the module is mounted to discharge heat and ensure that the internal temperature of the module is maintained between -30°C and 80°C. If the heat dissipation is poor, the integrated circuit may be abnormal or even burn out, causing the display system to fail to work properly.

The installation of the outdoor PDCharger modules must consider the requirements of lightning protection, waterproofing, moisture-proofing and heat dissipation. Only by making full preparations for these details can ensure that the module can operate stably and efficiently in various harsh environments and provide users with high-quality charging services.

External connectors placement

PDCharger-GSM (v.2.5):



X1: Input, input #1 (3.6-V to 40-V wide input operating voltage) :
Solar panel or external DC
Partnumber : DB2ERC-7.62-2P-GN

X2: Input, input #2 (3.6-V to 40-V wide input operating voltage) :
Solar panel or external DC
Partnumber : DB1ERC-5.08-2P-GN

X9: Output, From 2.5 V to 19.4V regulated output;
By default = 12V (factory settings) but depends on LiFePo battery output voltage
X9 output follows the battery if not in charging mode
Nominal = 12,8V ; Maximum = 14,4V ; Minimum = 10V ;
Partnumber : 2EDGRC-5.08-02P-14-100A

X41: Input , LiFePo 4 battery main cables connector (high current)
Partnumber : XT30PW-M

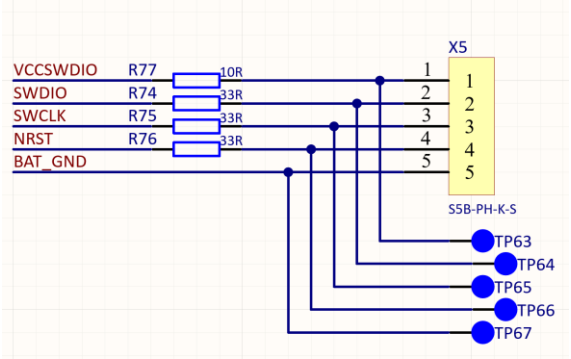
X7 : Input , NTC could be connected here (NTC 10k 1% 3905), for example THIS ONE
Partnumber : S2B-XH-A

X8 : Input , NTC from the **LiFePo battery Temp sensor must be connected** here (NTC 10k 1% 3905),

Partnumber : S2B-XH-A

X5: SWDIO STM32 programming connector

(Scheme part from the PCB side)



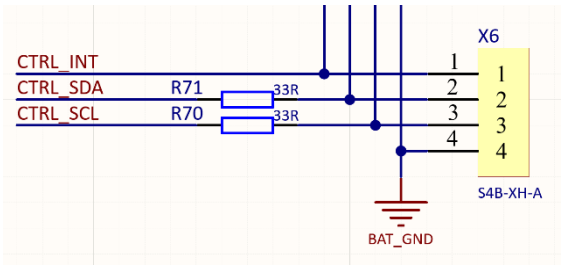
Partnumber : S5B-PH-K-S

X6: I2C EXT connector, for connecting external I2C devices for debug purposes

(Scheme part from the PCB side).

PDCharger v.2.2 must not have connected this I2C interface to the load side

PDCharger 2.3, 2.4 and 2.5 allows to connect this I2C interface to the load side



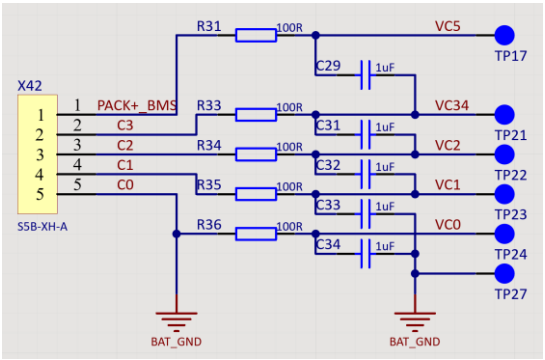
Attention!

The SDA and SCL lines on the Charger PCB have **10k** pull-ups and ~2,5V voltage level. Connection to Charger PCB via I2C should be aligned by driver strength and voltage level.

Partnumber : S4B-XH-A

X42: Input, balancer cable input from the LiFePo external battery

Partnumber : S5B-XH-A

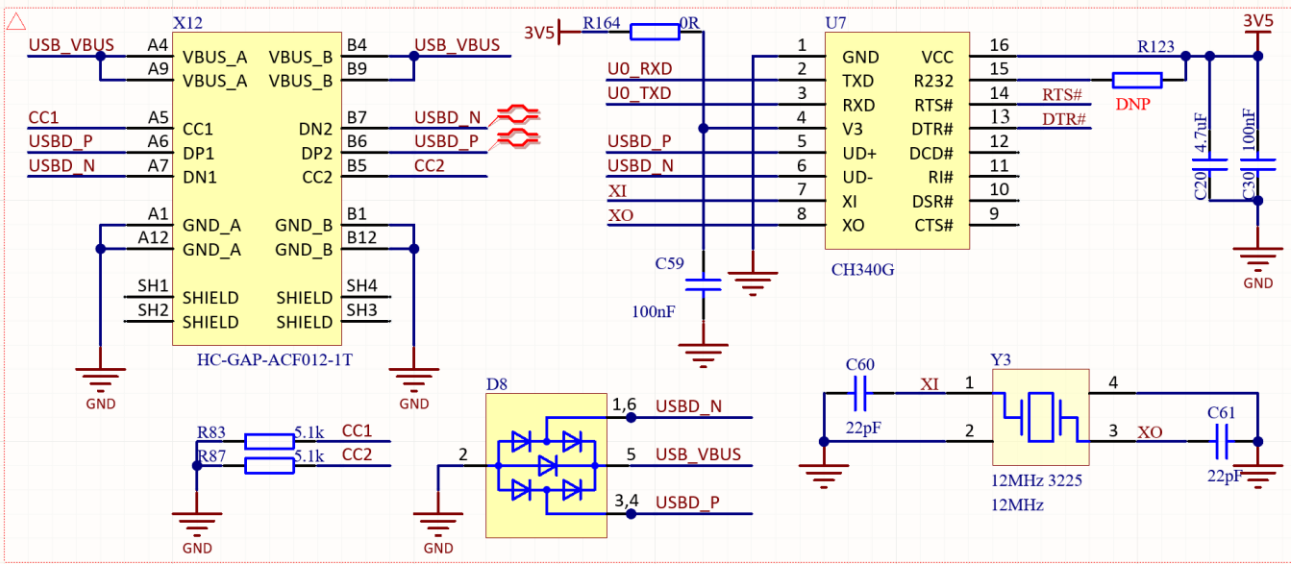


Battery balancer pinout	
XHP-05 connector pin	**connect to**
1 (Red wire)	+ 12.8V
2 (Black wire)	+ 9.6V
3 (Black wire)	+ 6.4V
4 (Black wire)	+ 3.2V
5 (Black wire)	GND

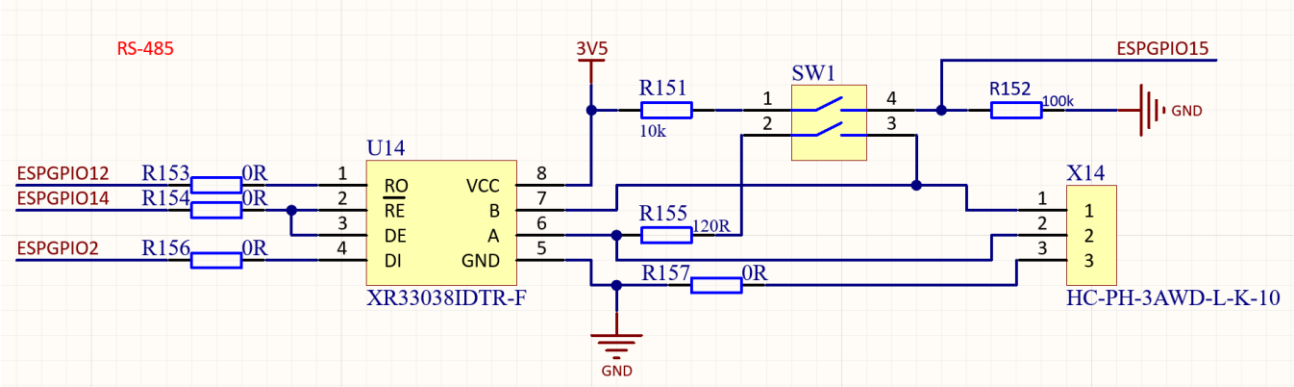
X3: Optional connector, could be not mounted. For I2C changing address use only

Partnumber : ZX-PZ2.54-2-5PZZ

X12: USB type C for ESP32 programming / IO, connected to the UART-USB converter onboard
Partnumber : HC-GAP-ACF012-1T

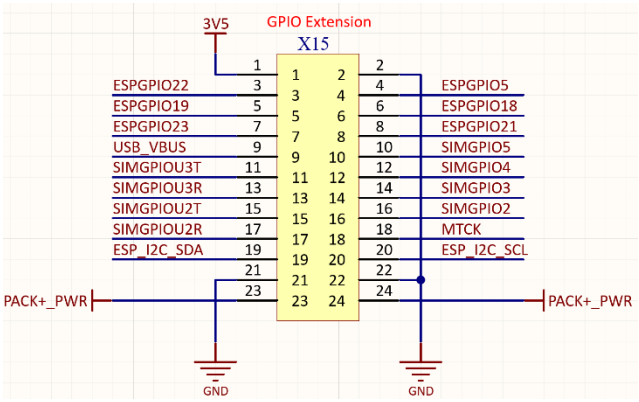


X14: RS485 bus connector, connected to the RS485 chip (XR33038IDTR-F) adapter onboard
Partnumber : HC-PH-3AWD-L-K-10



X4: SMA-type connector for WiFi / Bluetooth from ESP32 microcontroller
Partnumber : BWSMA-KWE-Z001

X15: GPIO extension connector from both ESP32 and SIM7080G subblocks
Partnumber : IDC 24 Pin, 321024MG0CBK00A01



Attention!

The "**PACK+_PWR**" is electrically connected to the Charger subblock out so there are +**12V** on those pins;

USB_VBUS is 5 Volts

3V5 is 3,5 Volts

X10: SMA-type connector for **GSM** channel

from Simcom SIM7080G module
Partnumber : BWSMA-KWE-Z001

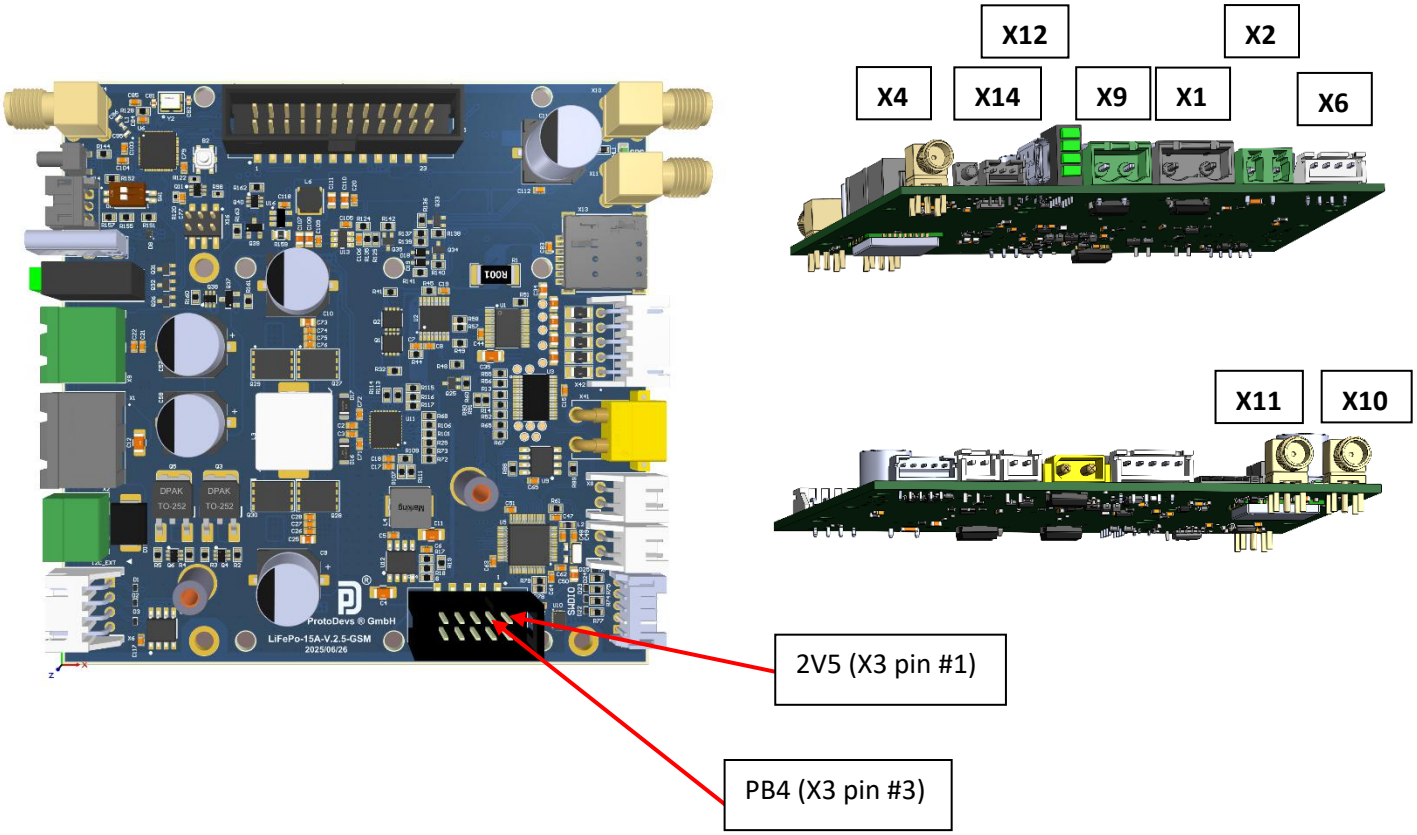
X11: SMA-type connector for **GPS** channel from Simcom SIM7080G module
Partnumber : BWSMA-KWE-Z001

X13: SIM card holder connector, connected to Simcom SIM7080G module
Partnumber : NANO SIM 7P H1.37 , SIM8066-6-1-14-01-A

The alternative function of X6 connector is **UART** protocol instead of **I2C**, this setting is controlled by PB4 state on **X3 GPIO** extender area, default interface is I2C (EXT_SDA, EXT_SCL).
In case of pulling up the **PB4** (10k resistor from PB4 to 2V5) to the high level, **SDA** becomes **TxD** and **SCL** becomes **RxD**. The PB4 check is executed every time the Charger PCB starts (powerup sequence).

More details you can find in the Charger API doc, please visit our forum for more details:

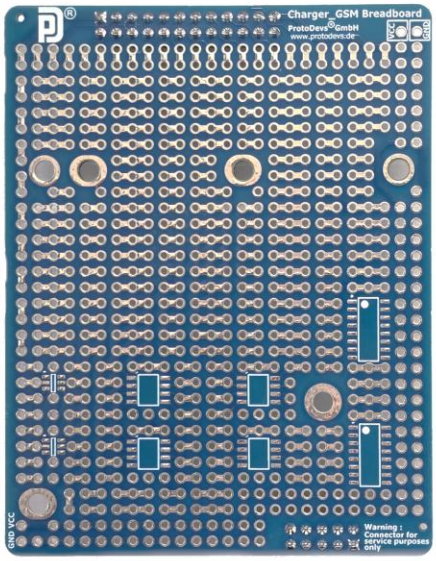
https://www.protodevs.de/forum/



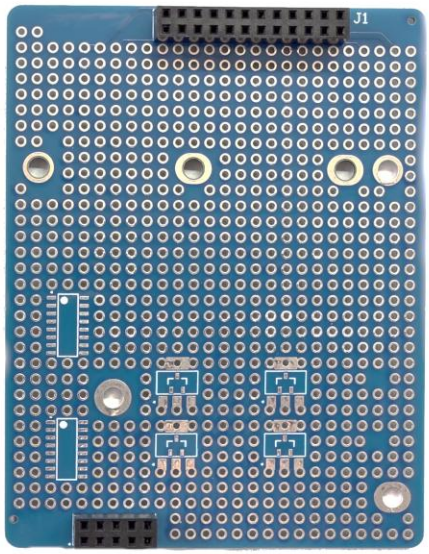
Add-on extension PCB for PD.Charger_GSM v.2.5

Extension breadboard PCB

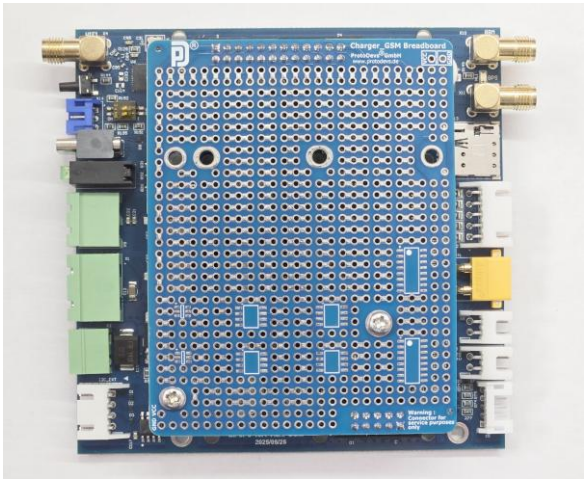
In case of need of a rapid prototyping of the additional or extended functions they could be implemented using already created and produced extension PCBs like on the picture.



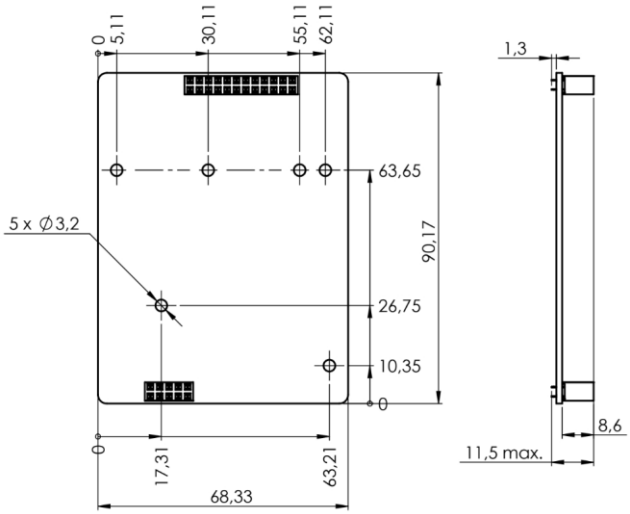
Breadboard PCB TOP side



Breadboard PCB BOTTOM side



PD.Charger_GSM v.2.5 with Breadboard PCB secured with 2x M3 screws



PD.Charger_GSM Breadboard PCB key dimensions

The **3D STEP models** of the breadboard PCBs could be found on the official ProtoDevs GmbH support page (http://protodevs.de/forum/). Custom design of extension PCBs could be made by ProtoDevs, please write to info@protodevs.de .

Preparations for PDCharger module communication

Board jumpers.
X3 port can be used for board configuration. Board reads jumpers during the startup.
Jumpers on pins PB0 and PB2 are used to change board i2c address.
Default address is 0x3B with jumpers you can increment the address on value from 0 to 0x10
- - pin not connected
0 - pin is LOW
1 - pin is HIGH

address increment: 0 2 4 6 8 A C E 10
pin PB0 state : - 0 1 - 0 1 - 0 1
pin PB2 state : - - - 0 0 0 1 1 1

Jumper on pin PB4 is used to specify communication format. default - i2c
format : I2C UART UART_HU
pin PB4 state: - 0 1

Communication formats.
I2C - used by default, tested at 100KHz speed, default board address is 0x3B.
You must know how many bytes you send, byte 0 in the response always contains the number of following bytes.
Connection:
SDA - X6.2,
SCL - X6.3,
GND - X6.4
UART - speed 115200. You must know how many bytes you send, byte 0 in the response always contains the number of following bytes.
All requests must contain one extra leading byte - number of following bytes. For example, command 00 01 in UART mode is 02 00 01,
where 02 is number of following bytes = 2.
Connection:
TxD - X6.2,
RxD - X6.3,
GND - X6.4
UART_HU - also uart, speed 115200, but in human-friendly format. you send/receive not bytes, but text representation of bytes separated with spaces.
Command is ended with ; character.
For example: 00 01; or 0 1;
Response in this format does not contain byte 0 and is ended with \n character.

List of commands supported (I2C)

Important note!

To preserve battery MCU sleeps most of the time. To start the communication X6.1 signal must be pulled LOW. Transition X6.1 from HIGH to LOW wakes up MCU, it stays awake for 5 seconds, communication must be started in 5 seconds after X6.1 changes from HIGH to LOW. MCU goes to sleep after 5 seconds after the last communication command.

Using CRC.

Crc is disabled by default. To enable crc command 10 01 01 must be issued.
When crc is enabled:

- Each request from host to board must be trailed with 4 extra bytes - crc hash, LSB first.
  For example, command to read if crc is used in requests/responses with disabled crc is
  00 01
  and with enabled crc same command is
  00 01 69 22 DE 36
  because crc32 hash of bytes 00 01 is 36DE2269
- Each response from boart to host, except the error responses, is also trailed with 4 extra bytes - crc hash, LCB first.
  CRC hash is calculated over the response bytes starting from 1! I.e. response byte 0 is not included in crc hash calculation.
  Response byte 0 is a number of the following response bytes. It is appended to the response after crc hash calculation,
  therefore is not included in crc calculation.
  Error response is always 00, it is not trailed with crc hash.
  For example, response with crc disabled is
  01 01
  Same response with crc enabled is
  05 01 1B DF 05 A5
  please note that crc hash A505DF1B is calculated over the 1 byte - 01. Byte 0 - 05 - is a number of the following bytes.

00 General information read.
Some information is unique, some duplicates charger/gauge register reads
response in case of command error: 00
Multibit numbers LSB first

00 - initialization status
request: 00 00
response: 01 BB
BB - 1 byte:
bit 0 = 1 - init error gauge,
bit 1 = 1 - gauge flash required update,
bit 2 = 1 - init error charger. charger state is updated on each iteration (every 5 seconds or so)
bit 3 = 1 - custom charger configuration was used

01 - use crc in requests/responses
request: 00 01
response: 01 BB
BB - 1 byte:
0 - crc not used,
1 - crc is used

02 - charger Input Voltage for Max Power Point detected
request: 00 02
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x1F_VAC_Max_Power_Point_Detected register

03 - charger MPPT status
request: 00 03
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x1A_MPPT_Control register

04 - gauge temperature
request: 00 04
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x08 Temperature() command

05 - gauge voltage
request: 00 05
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x09 Voltage() command

06 - gauge current
request: 00 06
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0A Current() command

07 - gauge average current
request: 00 07
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0B AverageCurrent() command

08 - gauge max error
request: 00 08
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0C MaxError() command

09 - gauge relative state of charge
request: 00 09
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0D RelativeStateOfCharge() command

0A .. 0E - gauge cell voltage. 0A - 0x3E CellVoltage1() ... 0E - 0x3A CellVoltage5()
request: 00 0A
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x3E CellVoltage1() ... 0x3A CellVoltage5() command

0F - gauge full charge capacity
request: 00 0F
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x10 FullChargeCapacity() command

10 - gauge charging current
request: 00 10
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x14 ChargingCurrent() command

11 - gauge charging voltage
request: 00 11
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x15 ChargingVoltage() command

12 - gauge battery status
request: 00 12
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x16 BatteryStatus() command

13 - gauge charging status
request: 00 13
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0055 ChargingStatus command

14 - gauge gauging status
request: 00 14

response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0056 GaugingStatus command

15 - gauge manufacturing status
request: 00 15
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0057 ManufacturingStatus command

16 - gauge AFE status
request: 00 16
response: 02 BBBB
BBBB - 2 bytes. Result of gauge 0x0058 AFEStatus command

17 - read flash settings
request: 00 17
response: 0E BBBB BBBB...
BBBB - 7 2-byte numbers, LSB first. First 4 numbers are reserved, numbers 4-7 can be used for user data:
0 - charger voltage threshold,
1 - input current for low voltage,
2 - input current for high voltage,
3 - configuration flags. bit 0 - enable MPPT in cycle, bit 1 - update input current in cycle

18 - read flash content
request: 00 18 AAAAAA CC
AAAAAA - 3 bytes. flash address. from 0 to 512 KBytes
CC - 1 byte. number of bytes to read. from 1 to 32
response: CC BB...
CC - 1 byte. Number of bytes in the response
BB... - CC bytes of flash content

19 - read custom charger registers configuration
request: 00 19
response: 1F BB...BB BBBB...BBBB
1F - 1 byte. Number of bytes in the response
BB...BB - 15 1-byte registers:
REG0x14_Precharge_and_Termination_Control,
REG0x15_Timer_Control,
REG0x16_ThreeStage_Charge_Control,
REG0x17_Charger_Control,
REG0x19_Power_Path_and_Reverse_Mode_Control,
REG0x1A_MPPT_Control,
REG0x1B_TS_Charging_Threshold_Control,
REG0x1C_TS_Charging_Region_Behavior_Control,
REG0x1D_TS_Reverse_Mode_Threshold_Control,
REG0x1E_Reverse_Undervoltage_Control,
REG0x2B_ADC_Control,

REG0x2C_ADC_Channel_Control,
REG0x3B_Gate_Driver_Strength_Control,
REG0x3C_Gate_Driver_Dead_Time_Control,
REG0x62_Reverse_Mode_Battery_Discharge_Current
BBBB...BBBB - 8 2-byte registers, LSB first:
REG0x00_Charge_Voltage_Limit,
REG0x02_Charge_Current_Limit,
REG0x06_Input_Current_DPM_Limit,
REG0x08_Input_Voltage_DPM_Limit,
REG0x0A_Reverse_Mode_Input_Current_Limit,
REG0x0C_Reverse_Mode_Input_Voltage_Limit,
REG0x10_Precharge_Current_Limit,
REG0x12_Termination_Current_Limit

1A - read MCU ticks since start. Normally 1 tick = 1 msec but ticks are stopped during sleep. This command can be used to find out if MCU restart happened since previous command
request: 00 19
response: BBBBBBBB
BBBBBBBB - 4 bytes. number of ticks since MCU start

1B - read firmware revision
request: 00 20
response: 0E DD...DD.BBBB
DD...DD - 11 bytes, build date in format 'MMM DD YYYY', ex: Sep 19 2024
. - 1 byte = '.'
BBBB - 2 bytes, build number byte in hex format

10 General information write
response in case of command error: 00
response in case of command success: 01 01

00 - restart
request: 10 00 BB
BB - 1 byte. 1 - restart

01 - set use/not use crc
request: 10 01 BB
BB - 1 byte. 0 - not use crc, 1 - use crc

02 - close/open gauge FETs
request: 10 02 BB
BB - 1 byte. 0 - close FETs, 1 - open FETs

03 - write flash settings
request: 10 03 BBBB BBBB...
BBBB - 7 2-byte numbers, LSB first. First 4 numbers are reserved, numbers 4-7 can be used

for user data:
0 - charger voltage threshold,
1 - input current for low voltage,
2 - input current for high voltage,
3 - configuration flags. bit 0 - enable MPPT in cycle, bit 1 - update input current in cycle

04 - write flash content. Writes arbitrary content to flash. Up to 32 bytes. Address must be 0x2000 or more. Just writes, does not erase.
request: 10 04 AAAAAA CC BB...
AAAAAA - 3 bytes. flash address. from 0x2000 to 0x80000
CC - 1 byte. number of bytes to write. from 1 to 32
BB... - CC bytes to write to flash content

05 - erase flash sector. Erases flash sector. Sector size is 4096bytes, sectors 2 to 127 are allowed for user use
request: 10 05 AAAAAA
AAAAAA - 3 bytes. flash sector address. 0..4095 - sector 0, 4096..8191 - sector 1, etc.

06 - write custom charger registers configuration. these register values will be loaded to charger on the next charger start
request: 10 06 BB...BB BBBB...BBBB
BB...BB - 15 1-byte registers:
REG0x14_Precharge_and_Termination_Control,
REG0x15_Timer_Control,
REG0x16_ThreeStage_Charge_Control,
REG0x17_Charger_Control,
REG0x19_Power_Path_and_Reverse_Mode_Control,
REG0x1A_MPPT_Control,
REG0x1B_TS_Charging_Threshold_Control,
REG0x1C_TS_Charging_Region_Behavior_Control,
REG0x1D_TS_Reverse_Mode_Threshold_Control,
REG0x1E_Reverse_Undervoltage_Control,
REG0x2B_ADC_Control,
REG0x2C_ADC_Channel_Control,
REG0x3B_Gate_Driver_Strength_Control,
REG0x3C_Gate_Driver_Dead_Time_Control,
REG0x62_Reverse_Mode_Battery_Discharge_Current
BBBB...BBBB - 8 2-byte registers, LSB first:
REG0x00_Charge_Voltage_Limit,
REG0x02_Charge_Current_Limit,
REG0x06_Input_Current_DPM_Limit,
REG0x08_Input_Voltage_DPM_Limit,
REG0x0A_Reverse_Mode_Input_Current_Limit,
REG0x0C_Reverse_Mode_Input_Voltage_Limit,
REG0x10_Precharge_Current_Limit,
REG0x12_Termination_Current_Limit
special request: 10 06 FF - erases custom charger registers configuration and makes charger

use the default configuration

07 - write gauge balancing configuration. Flash address 0x460C. After flash is updated gauge must be restarted for settings to be applied. Since this setting is in flash it remains unchanged after the restart or power reconnect.

request: 10 07 BB RR

BB - 1 byte. Content of flash address 0x460C. 0 - disable cell balancing, 1 - enable cell balancing

RR - 1 byte. 1 - restart gauge to apply new settings

08 - write battery capacity to gauge, adjusts related settings accordingly - design capacity, learned FCC, newr full, etc. After flash is updated gauge must be restarted for settings to be applied.

request: 10 08 BBBB RR

BBBB - 2 bytes. Battery design capacity value, LSB first. Allowed range 1000 - 32767

RR - 1 byte. 1 - restart gauge to apply new settings

01 Charger registers read

response in case of command error: 00

Multibit numbers LSB first

00 - REG0x00_Charge_Voltage_Limit Register

request: 01 00

response: 02 BBBB

BBBB - 2 bytes. Content of charger REG0x00_Charge_Voltage_Limit Register

02 - REG0x02_Charge_Current_Limit Register

request: 01 02

response: 02 BBBB

BBBB - 2 bytes. Content of charger REG0x02_Charge_Current_Limit Register

06 - REG0x06_Input_Current_DPM_Limit Register

request: 01 06

response: 02 BBBB

BBBB - 2 bytes. Content of charger REG0x06_Input_Current_DPM_Limit Register

08 - REG0x08_Input_Voltage_DPM_Limit Register

request: 01 08

response: 02 BBBB

BBBB - 2 bytes. Content of charger REG0x08_Input_Voltage_DPM_Limit Register

0A - REG0x0A_Reverse_Mode_Input_Current_Limit Register

request: 01 0A

response: 02 BBBB

BBBB - 2 bytes. Content of charger REG0x0A_Reverse_Mode_Input_Current_Limit Register

0C - REG0x0C_Reverse_Mode_Input_Voltage_Limit Register

request: 01 0C
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x0C_Reverse_Mode_Input_Voltage_Limit Register

10 - REG0x10_Precharge_Current_Limit Register
request: 01 10
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x10_Precharge_Current_Limit Register

12 - REG0x12_Termination_Current_Limit Register
request: 01 12
response: 02 BBBB
BBBB - 2 byte. Content of charger REG0x12_Termination_Current_Limit Register

14 - REG0x14_Precharge_and_Termination_Control Register
request: 01 14
response: 01 BB
BB - 1 byte. Content of charger REG0x14_Precharge_and_Termination_Control Register

15 - REG0x15_Timer_Control Register
request: 01 15
response: 01 BB
BB - 1 byte. Content of charger REG0x15_Timer_Control Register

16 - REG0x16_ThreeStage_Charge_Control Register
request: 01 16
response: 01 BB
BB - 1 byte. Content of charger REG0x16_ThreeStage_Charge_Control Register

17 - REG0x17_Charger_Control Register
request: 01 17
response: 01 BB
BB - 1 byte. Content of charger REG0x17_Charger_Control Register

18 - REG0x18_Pin_Control Register
request: 01 18
response: 01 BB
BB - 1 byte. Content of charger REG0x18_Pin_Control Register

19 - REG0x19_Power_Path_and_Reverse_Mode_Control Register
request: 01 19
response: 01 BB
BB - 1 byte. Content of charger REG0x19_Power_Path_and_Reverse_Mode_Control Register

1A - REG0x1A_MPPT_Control Register
request: 01 1A
response: 01 BB

BB - 1 byte. Content of charger REG0x1A_MPPT_Control Register

1B - REG0x1B_TS_Charging_Threshold_Control Register
request: 01 1B
response: 01 BB
BB - 1 byte. Content of charger REG0x1B_TS_Charging_Threshold_Control Register

1C - REG0x1C_TS_Charging_Region_Behavior_Control Register
request: 01 1C
response: 01 BB
BB - 1 byte. Content of charger REG0x1C_TS_Charging_Region_Behavior_Control Register

1D - REG0x1D_TS_Reverse_Mode_Threshold_Control Register
request: 01 1D
response: 01 BB
BB - 1 byte. Content of charger REG0x1D_TS_Reverse_Mode_Threshold_Control Register

1E - REG0x1E_Reverse_Undervoltage_Control Register
request: 01 1E
response: 01 BB
BB - 1 byte. Content of charger REG0x1E_Reverse_Undervoltage_Control Register

1F - REG0x1F_VAC_Max_Power_Point_Detected Register
request: 01 1F
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x1F_VAC_Max_Power_Point_Detected Register

21 - REG0x21_Charger_Status_1 Register
request: 01 21
response: 01 BB
BB - 1 byte. Content of charger REG0x21_Charger_Status_1 Register

22 - REG0x22_Charger_Status_2 Register
request: 01 22
response: 01 BB
BB - 1 byte. Content of charger REG0x22_Charger_Status_2 Register

23 - REG0x23_Charger_Status_3 Register
request: 01 23
response: 01 BB
BB - 1 byte. Content of charger REG0x23_Charger_Status_3 Register

24 - REG0x24_Fault_Status Register
request: 01 24
response: 01 BB
BB - 1 byte. Content of charger REG0x24_Fault_Status Register

25 - REG0x25_Charger_Flag_1 Register
request: 01 25
response: 01 BB
BB - 1 byte. Content of charger REG0x25_Charger_Flag_1 Register

26 - REG0x26_Charger_Flag_2 Register
request: 01 26
response: 01 BB
BB - 1 byte. Content of charger REG0x26_Charger_Flag_2 Register

27 - REG0x27_Fault_Flag Register
request: 01 27
response: 01 BB
BB - 1 byte. Content of charger REG0x27_Fault_Flag Register

28 - REG0x28_Charger_Mask_1 Register
request: 01 28
response: 01 BB
BB - 1 byte. Content of charger REG0x28_Charger_Mask_1 Register

29 - REG0x29_Charger_Mask_2 Register
request: 01 29
response: 01 BB
BB - 1 byte. Content of charger REG0x29_Charger_Mask_2 Register

2A - REG0x2A_Fault_Mask Register
request: 01 2A
response: 01 BB
BB - 1 byte. Content of charger REG0x2A_Fault_Mask Register

2B - REG0x2B_ADC_Control Register
request: 01 2B
response: 01 BB
BB - 1 byte. Content of charger REG0x2B_ADC_Control Register

2C - REG0x2C_ADC_Channel_Control Register
request: 01 2C
response: 01 BB
BB - 1 byte. Content of charger REG0x2C_ADC_Channel_Control Register

2D - REG0x2D_IAC_ADC Register
request: 01 2D
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x2D_IAC_ADC Register

2F - REG0x2F_IBAT_ADC Register
request: 01 2F

response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x2F_IBAT_ADC Register

31 - REG0x31_VAC_ADC Register
request: 01 31
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x31_VAC_ADC Register

33 - REG0x33_VBAT_ADC Register
request: 01 33
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x33_VBAT_ADC Register

37 - REG0x37_TS_ADC Register
request: 01 37
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x37_TS_ADC Register

39 - REG0x39_VFB_ADC Register
request: 01 39
response: 02 BBBB
BBBB - 2 bytes. Content of charger REG0x39_VFB_ADC Register

3B - REG0x3B_Gate_Driver_Strength_Control Register
request: 01 3B
response: 01 BB
BB - 1 byte. Content of charger REG0x3B_Gate_Driver_Strength_Control Register

3C - REG0x3C_Gate_Driver_Dead_Time_Control Register
request: 01 3C
response: 01 BB
BB - 1 byte. Content of charger REG0x3C_Gate_Driver_Dead_Time_Control Register

3D - REG0x3D_Part_Information Register
request: 01 3D
response: 01 BB
BB - 1 byte. Content of charger REG0x3D_Part_Information Register

62 - REG0x62_Reverse_Mode_Battery_Discharge_Current Register
request: 01 62
response: 01 BB
BB - 1 byte. Content of charger REG0x62_Reverse_Mode_Battery_Discharge_Current Register

11 Charger registers write

response in case of command error: 00
response in case of command success: 01 01
Multibit numbers LSB first

00 - REG0x00_Charge_Voltage_Limit Register
request: 11 00 BBBB
BBBB - 2 bytes. New content of charger REG0x00_Charge_Voltage_Limit Register

02 - REG0x02_Charge_Current_Limit Register
request: 11 02 BBBB
BBBB - 2 bytes. New content of charger REG0x02_Charge_Current_Limit Register

06 - REG0x06_Input_Current_DPM_Limit Register
request: 11 06 BBBB
BBBB - 2 bytes. New content of charger REG0x06_Input_Current_DPM_Limit Register

08 - REG0x08_Input_Voltage_DPM_Limit Register
request: 11 08 BBBB
BBBB - 2 bytes. New content of charger REG0x08_Input_Voltage_DPM_Limit Register

0A - REG0x0A_Reverse_Mode_Input_Current_Limit Register
request: 11 0A BBBB
BBBB - 2 bytes. New content of charger REG0x0A_Reverse_Mode_Input_Current_Limit Register

0C - REG0x0C_Reverse_Mode_Input_Voltage_Limit Register
request: 11 0C BBBB
BBBB - 2 bytes. New content of charger REG0x0C_Reverse_Mode_Input_Voltage_Limit Register

10 - REG0x10_Precharge_Current_Limit Register
request: 11 10 BBBB
BBBB - 2 bytes. New content of charger REG0x10_Precharge_Current_Limit Register

12 - REG0x12_Termination_Current_Limit Register
request: 11 12 BBBB
BBBB - 2 bytes. New content of charger REG0x12_Termination_Current_Limit Register

14 - REG0x14_Precharge_and_Termination_Control Register
request: 11 14 BB
BB - 1 byte. New content of charger REG0x14_Precharge_and_Termination_Control Register

15 - REG0x15_Timer_Control Register
request: 11 15 BB
BB - 1 byte. New content of charger REG0x15_Timer_Control Register

16 - REG0x16_ThreeStage_Charge_Control Register

request: 11 16 BB
BB - 1 byte. New content of charger REG0x16_ThreeStage_Charge_Control Register

17 - REG0x17_Charger_Control Register
request: 11 17 BB
BB - 1 byte. New content of charger REG0x17_Charger_Control Register

18 - REG0x18_Pin_Control Register
request: 11 18 BB
BB - 1 byte. New content of charger REG0x18_Pin_Control Register

19 - REG0x19_Power_Path_and_Reverse_Mode_Control Register
request: 11 19 BB
BB - 1 byte. New content of charger REG0x19_Power_Path_and_Reverse_Mode_Control Register

1A - REG0x1A_MPPT_Control Register
request: 11 1A BB
BB - 1 byte. New content of charger REG0x1A_MPPT_Control Register

1B - REG0x1B_TS_Charging_Threshold_Control Register
request: 11 1B BB
BB - 1 byte. New content of charger REG0x1B_TS_Charging_Threshold_Control Register

1C - REG0x1C_TS_Charging_Region_Behavior_Control Register
request: 11 1C BB
BB - 1 byte. New content of charger REG0x1C_TS_Charging_Region_Behavior_Control Register

1D - REG0x1D_TS_Reverse_Mode_Threshold_Control Register
request: 11 1D BB
BB - 1 byte. New content of charger REG0x1D_TS_Reverse_Mode_Threshold_Control Register

1E - REG0x1E_Reverse_Undervoltage_Control Register
request: 11 1E BB
BB - 1 byte. New content of charger REG0x1E_Reverse_Undervoltage_Control Register

28 - REG0x28_Charger_Mask_1 Register
request: 11 28 BB
BB - 1 byte. New content of charger REG0x28_Charger_Mask_1 Register

29 - REG0x29_Charger_Mask_2 Register
request: 11 29 BB
BB - 1 byte. New content of charger REG0x29_Charger_Mask_2 Register

2A - REG0x2A_Fault_Mask Register
request: 11 2A BB
BB - 1 byte. New content of charger REG0x2A_Fault_Mask Register

2B - REG0x2B_ADC_Control Register
request: 11 2B BB
BB - 1 byte. New content of charger REG0x2B_ADC_Control Register

2C - REG0x2C_ADC_Channel_Control Register
request: 11 2C BB
BB - 1 byte. New content of charger REG0x2C_ADC_Channel_Control Register

3B - REG0x3B_Gate_Driver_Strength_Control Register
request: 11 3B BB
BB - 1 byte. New content of charger REG0x3B_Gate_Driver_Strength_Control Register

3C - REG0x3C_Gate_Driver_Dead_Time_Control Register
request: 11 3C BB
BB - 1 byte. New content of charger REG0x3C_Gate_Driver_Dead_Time_Control Register

62 - REG0x62_Reverse_Mode_Battery_Discharge_Current Register
request: 11 62 BB
BB - 1 byte. New content of charger REG0x62_Reverse_Mode_Battery_Discharge_Current Register

04 Gauge ManufacturerAccess read
response in case of command error: 00
Multibit numbers LSB first

0001 - 0x0001 Device Type
request: 04 01 00
response: CC bbb...
CC - 1 byte. Number of following bytes
bbb... - byte(0) ... byte(CC-1)

0002 - 0x0002 Firmware Version
0003 - 0x0003 Hardware Version
0004 - 0x0004 Instruction Flash Signature
0005 - 0x0005 Static DF Signature
0006 - 0x0006 Chemical ID
0008 - 0x0008 Static Chem DF Signature
0050 - 0x0050 SafetyAlert
0051 - 0x0051 SafetyStatus
0052 - 0x0052 PFAlert
0053 - 0x0053 PFStatus
0054 - 0x0054 OperationStatus
0055 - 0x0055 ChargingStatus
0056 - 0x0056 GaugingStatus
0057 - 0x0057 ManufacturingStatus
0058 - 0x0058 AFEStatus

0059 - 0x0059 AFEConfig
005A - 0x005A AFEVCx
005B - 0x005B AFEData
0060 - 0x0060 Lifetime Data Block 1
0061 - 0x0061 Lifetime Data Block 2
0062 - 0x0062 Lifetime Data Block 3
0063 - 0x0063 Lifetime Data Block 4
0064 - 0x0064 Lifetime Data Block 5
0065 - 0x0065 Lifetime Data Block 6
0066 - 0x0066 Lifetime Data Block 7
0070 - 0x0070 ManufacturerInfo
0071 - 0x0071 DAStatus1
0072 - 0x0072 DAStatus2
0080 - 0x0080 CUV Snapshot
0081 - 0x0081 COV Snapshot

14 Gauge ManufacturerAccess write
response in case of command error: 00
response in case of command success: 01 01
Multibit numbers LSB first

0010 - 0x0010 SHUTDOWN Mode - Execute SHUTDOWN Mode command
request: 14 10 00

0011 - 0x0011 SLEEP Mode
001B - 0x001B Cell Balance Toggle
001C - 0x001C AFE Delay Disable
001D - 0x001D SAFE Toggle
001E - 0x001E PRE-CHG FET
001F - 0x001F CHG FET
0020 - 0x0020 DSG FET
0022 - 0x0022 FET Control
0023 - 0x0023 Lifetime Data Collection
0024 - 0x0024 Permanent Failure
0025 - 0x0025 Black Box Recorder
0026 - 0x0026 SAFE
0027 - 0x0027 LED Display Enable
0028 - 0x0028 Lifetime Data Reset
0029 - 0x0029 Permanent Fail Data Reset
002A - 0x002A Black Box Recorder Reset
002B - 0x002B LED TOGGLE
002C - 0x002C LED Display Press
002D - 0x002D CALIBRATION Mode
0041 - 0x0041 Device Reset
0F00 - 0x0F00 ROM Mode
F080 - 0xF080 Exit Calibration Output Mode

F081 - 0xF081 OutputCellVoltageforCalibration
F082 - 0xF082 OutputCellVoltageCCandTempforCalibration

05 Gauge SBS Command read

response in case of command error: 00
Multibit numbers LSB first

01 - 0x01 RemainingCapacityAlarm()
request: 05 01
response: CC bbb...
CC - 1 byte. Number of following bytes
bbb... - byte(0) ... byte(CC-1)

02 - 0x02 RemainingTimeAlarm()
03 - 0x03 BatteryMode()
04 - 0x04 AtRate()
05 - 0x05 AtRateTimeToFull()
06 - 0x06 AtRateTimeToEmpty()
07 - 0x07 AtRateOK()
08 - 0x08 Temperature()
09 - 0x09 Voltage()
0A - 0x0A Current()
0B - 0x0B AverageCurrent()
0C - 0x0C MaxError()
0D - 0x0D RelativeStateOfCharge()
0E - 0x0E AbsoluteStateOfCharge()
0F - 0x0F RemainingCapacity()
10 - 0x10 FullChargeCapacity()
11 - 0x11 RunTimeToEmpty()
12 - 0x12 AverageTimeToEmpty()
13 - 0x13 AverageTimeToFull()
14 - 0x14 ChargingCurrent()
15 - 0x15 ChargingVoltage()
16 - 0x16 BatteryStatus()
17 - 0x17 CycleCount()
18 - 0x18 DesignCapacity()
19 - 0x19 DesignVoltage()
1A - 0x1A SpecificationInfo()
1B - 0x1B ManufacturerDate()
1C - 0x1C SerialNumber()
20 - 0x20 ManufacturerName()
21 - 0x21 DeviceName()
22 - 0x22 DeviceChemistry()
23 - 0x23 ManufacturerData()/CalibrationData()
2B - 0x2B HostFETControl
2C - 0x2C GPIOStatus

2D - 0x2D GPIOControl
2E - 0x2E VAUXVoltage()
3B - 0x3B CellVoltage5
3C - 0x3C CellVoltage4
3D - 0x3D CellVoltage3
3E - 0x3E CellVoltage2
3F - 0x3F CellVoltage1
4C - 0x4C DynamicPower()
4D - 0x4D ExtAveCellVoltage()
4E - 0x4E PendingEDV()
4F - 0x4F StateOfHealth (SOH)
50 - 0x50 SafetyAlert
51 - 0x51 SafetyStatus
52 - 0x52 PFAlert
53 - 0x53 PFStatus
54 - 0x54 OperationStatus
55 - 0x55 ChargingStatus
56 - 0x56 GaugingStatus
57 - 0x57 ManufacturingStatus
58 - 0x58 AFEStatus
59 - 0x59 AFEConfig
5A - 0x5A AFEVCx
5B - 0x5B AFEData
60 - 0x60 Lifetime Data Block 1
61 - 0x61 Lifetime Data Block 2
62 - 0x62 Lifetime Data Block 3
63 - 0x63 Lifetime Data Block 4
64 - 0x64 Lifetime Data Block 5
65 - 0x65 Lifetime Data Block 6
66 - 0x66 Lifetime Data Block 7
70 - 0x70 ManufacturerInfo
71 - 0x71 DAStatus1
72 - 0x72 DAStatus2
80 - 0x80 CUV Snapshot
81 - 0x81 COV Snapshot

15 Gauge SBS Command write

response in case of command error: 00
response in case of command success: 01 01
Multibit numbers LSB first

01 - 0x01 RemainingCapacityAlarm() - Set new value of RemainingCapacityAlarm
request: 15 01 BBBB
BBBB - 2 bytes. New value

02 - 0x02 RemainingTimeAlarm()
03 - 0x03 BatteryMode()
04 - 0x04 AtRate()
17 - 0x17 CycleCount()
18 - 0x18 DesignCapacity()
19 - 0x19 DesignVoltage()
1A - 0x1A SpecificationInfo()
1B - 0x1B ManufacturerDate()
1C - 0x1C SerialNumber()
2B - 0x2B HostFETControl
2D - 0x2D GPIOControl

06 Gauge flash read

response in case of command error: 00
Flash address - LSB first

request: 06 AAAA
AAAA - 2 bytes. Flash address, must be in range 0x4000–0x5FFF
response: CCbbb...
CC - 1 byte. Number of following bytes
bbb... - byte(0) ... byte(CC-1)

16 Gauge flash write

response in case of command error: 00
response in case of command success: 01 01
Flash address - LSB first

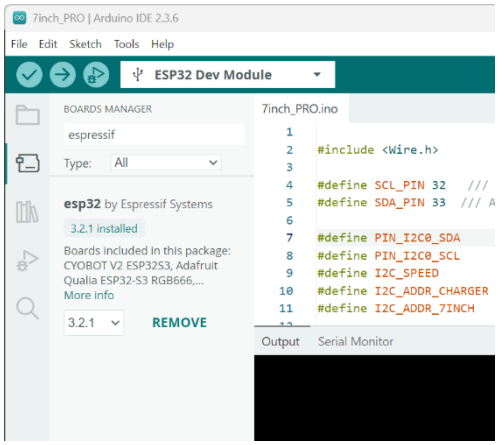
request: 06 AAAA CC bbb...
AAAA - 2 bytes. Flash address, must be in range 0x4000–0x5FFF
CC - 1 byte. Number of following bytes
bbb... - byte(0) ... byte(CC-1)

Arduino examples for onboard ESP32

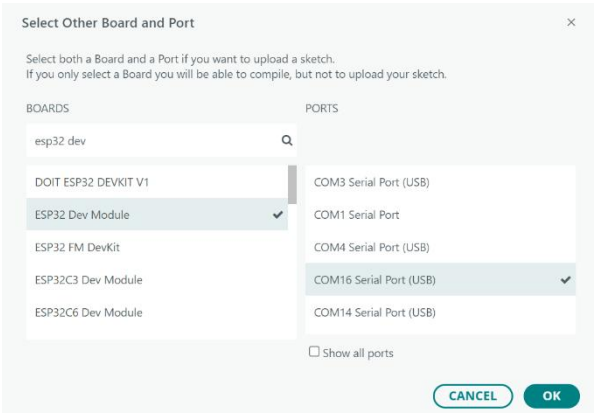
For the demonstration and reference purposes, several small examples would be shown.
The Arduino IDE settings must be set accordingly in order to compile and run the examples successfully.

Arduino IDE settings

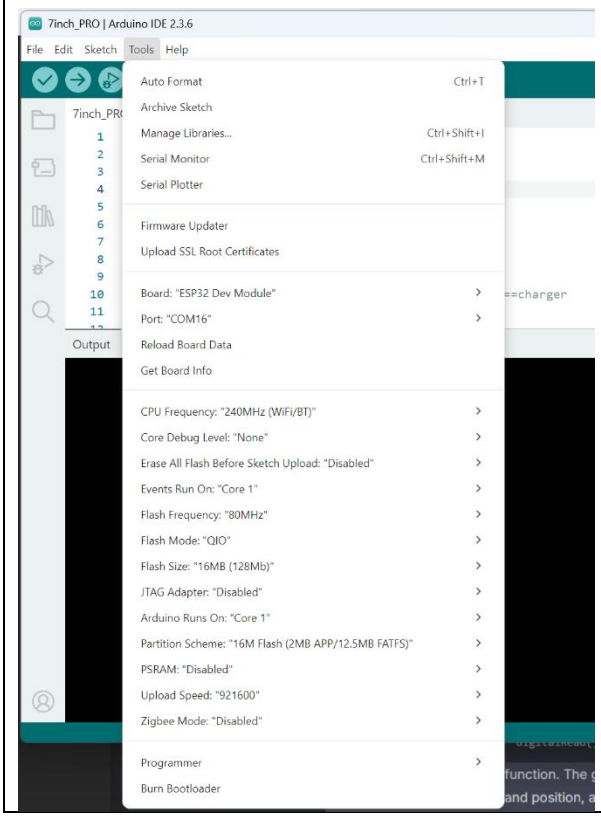
1. The Espressif "esp32" library must be installed:



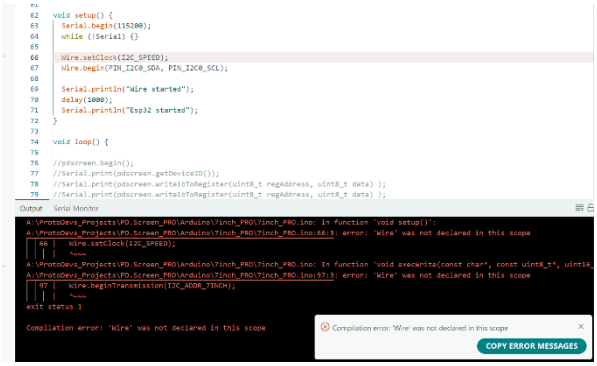
2. The optimal preconfigured board should be set as "ESP32 Dev Module"



3. The setting should be set as following



4. Install all the needed libraries if the compilation shows errors



5. Change the code as you like and experiment with the outputs.
Have a nice coding!

Arduino example #1 : controlling PD.Screen with I2C commands

```
#include <Wire.h>

#define SCL_PIN 32   /// A5 is hardware SCL on Uno
#define SDA_PIN 33  /// A4 is hardware SDA on Uno

#define PIN_I2C0_SDA     32//5//0
#define PIN_I2C0_SCL     33//4//1
#define I2C_SPEED        100000
#define I2C_ADDR_CHARGER      0x3B// 0x3B – default I2C address for PD.Charger and PD.Charger_GSM
#define I2C_ADDR_7INCH        0x39// 0x39 – default I2C address for PD.Screen 7" m30

#define I2C_START_ADDRESS 0x08
#define I2C_END_ADDRESS   0x77


static const char* serial_command_flashinfo     = "flashinfo;";
static const char* serial_sector_write          = "wrsect;";
static const char* serial_sector_read           = "rdsect;";
static const char* serial_pictures_recache      = "recache;";
static const char* serial_draw_picture          = "draw_pic;";
static const char* serial_draw_picture2         = "draw_pic2;";
static const char* serial_draw_picture_op       = "draw_pic_op;";
static const char* serial_draw_rect             = "draw_rect;";
static const char* serial_fill_rect             = "fill_rect;";
static const char* serial_draw_line             = "draw_line;";
static const char* serial_draw_circle           = "draw_circ;";
static const char* serial_fill_circle           = "fill_circ;";
static const char* serial_draw_triangle         = "draw_tria;";
static const char* serial_fill_triangle         = "fill_tria;";
static const char* serial_draw_roundrect        = "draw_rrect;";
static const char* serial_fill_roundrect        = "fill_rrect;";
static const char* serial_draw_pixel            = "draw_pixel;";
static const char* serial_draw_text             = "draw_text;";

static const char* serial_get_temp              = "get_temp;";
static const char* serial_get_light             = "get_light;";
static const char* serial_get_time              = "get_time;";
static const char* serial_set_time              = "set_time;";
static const char* serial_get_rtcram            = "get_rtcram;";
static const char* serial_set_rtcram            = "set_rtcram;";
static const char* serial_get_touch             = "get_touch;";
static const char* serial_touchcapture_start    = "touch_capton;";
static const char* serial_touchcapture_stop     = "touch_captoff;";
static const char* serial_get_capturedline      = "get_capturedline;";
```



```
static const char* serial_tft_on               = "tft_on;";
static const char* serial_tft_off              = "tft_off;";
static const char* serial_screen_on            = "screen_on;";
static const char* serial_screen_off           = "screen_off;";
static const char* serial_screen_bklight       = "screen_bkl;";

static const char* serial_command_confirmation  = "ok\n";
const unsigned long delayTime = 100;
unsigned long lastmillis;

void setup() {
  Serial.begin(115200);
  while (!Serial) {}
  Wire.setClock(I2C_SPEED);
  Wire.begin(PIN_I2C0_SDA, PIN_I2C0_SCL);
  delay(1000);
  Serial.println("Esp32 started");
}

void loop() {
  screen_off();
  delay(100);
  tft_on();
  fill_rect(1, 0, 0, 800, 480, 0xFF000000);
  delay(3);
  screen_bkl(7);
  screen_on();
  draw_text(1, 20, 130, 0xFFFF, 0xFF8888FF, 1, 3, "PD.Screen project");
  delay(3000);
}

void execWrite(const char* command, const uint8_t* params, const uint16_t params_cnt) {
  Serial.println(command);
  Wire.beginTransmission(I2C_ADDR_7INCH);
  //Wire.write(command);
  Wire.write((const uint8_t*)command, strlen(command));
  Wire.write(params, params_cnt);
  Wire.endTransmission();
  int16_t cnt = strlen(serial_command_confirmation);
  int16_t vi = Wire.requestFrom(I2C_ADDR_7INCH, cnt);
  char buff[64];
  Wire.readBytes(buff, vi);
  buff[vi] = 0;
  if ((vi != cnt) || (strcmp(buff, serial_command_confirmation) != 0)) {
    Serial.printf("Write failed! Received %d bytes: %s\n", vi, buff);
  }
}
```



```
void draw_text(const uint8_t idx, const uint16_t x, const uint16_t y, const uint32_t clr, const uint32_t
bg_clr,
 const uint16_t font, const uint8_t sz, const char* txt) {
  uint8_t params[64];
  int16_t params_idx = 0;
  byte2params(idx, params, params_idx);
  word2params(x, params, params_idx);
  word2params(y, params, params_idx);
  dword2params(clr, params, params_idx);
  dword2params(bg_clr, params, params_idx);
  word2params(font, params, params_idx);
  byte2params(sz, params, params_idx);
  uint8_t cnt = strlen(txt);
  byte2params(cnt, params, params_idx);
  char* c = (char*)txt;
  while (*c != 0) {
    params[params_idx++] = *c;
    c++;
  }
  execWrite(serial_draw_text, params, params_idx);
}

void screen_off() {
  execWrite(serial_screen_off, nullptr, 0);
}

void screen_on() {
  execWrite(serial_screen_on, nullptr, 0);
}

void tft_off() {
  execWrite(serial_tft_off, nullptr, 0);
}

void tft_on() {
  execWrite(serial_tft_on, nullptr, 0);
}

void screen_bkl(const uint8_t val) {
  execWrite(serial_screen_bklight, &val, 1);
}
```

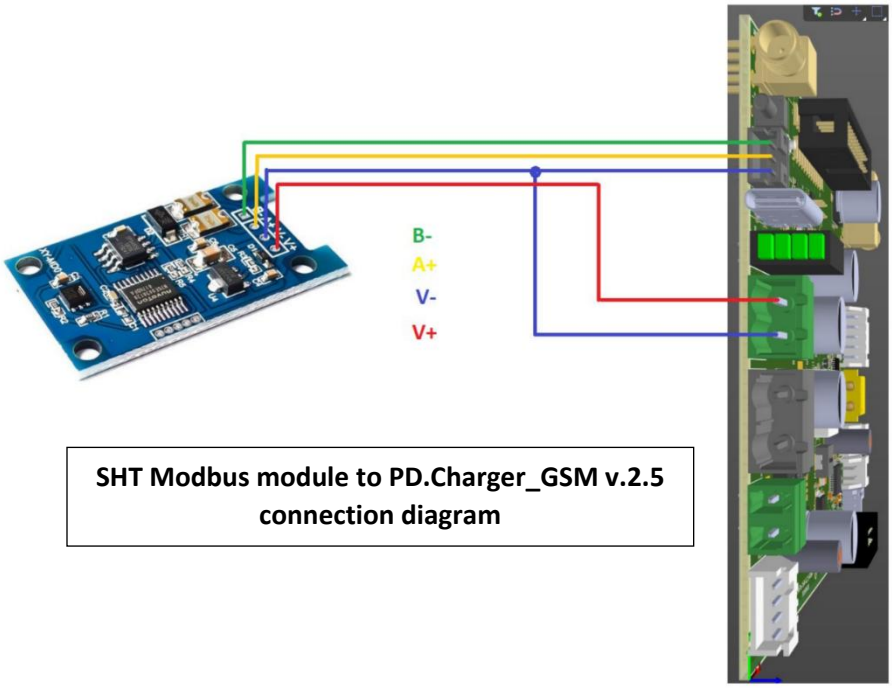
```
void fill_rect(const uint8_t layer, const uint16_t x, const uint16_t y, const uint16_t w, const uint16_t h,
```



```
const uint32_t clr) {
  uint8_t params[13];
  int16_t params_idx = 0;
  byte2params(layer, params, params_idx);
  word2params(x, params, params_idx);
  word2params(y, params, params_idx);
  word2params(w, params, params_idx);
  word2params(h, params, params_idx);
  dword2params(clr, params, params_idx);
  execWrite(serial_fill_rect, params, params_idx);
}

void byte2params(const uint8_t val, uint8_t* params, int16_t& params_idx) {
  params[params_idx++] = val;
}

void word2params(const uint16_t val, uint8_t* params, int16_t& params_idx) {
  params[params_idx++] = val;
  params[params_idx++] = val >> 8;
}

void dword2params(const uint32_t val, uint8_t* params, int16_t& params_idx) {
  params[params_idx++] = val;
  params[params_idx++] = val >> 8;
  params[params_idx++] = val >> 16;
  params[params_idx++] = val >> 24;
}

uint16_t bytes2word(const uint8_t* params, int16_t params_idx) {
  return params[params_idx++] | (params[params_idx++] << 8);
}

uint32_t bytes2dword(const uint8_t* params, int16_t params_idx) {
  return params[params_idx++] | (params[params_idx++] << 8) | (params[params_idx++] << 16) |
(params[params_idx++] << 24);
}
```


Modbus RTU temperature and humidity sensor read (SHT20)

```
#include <ModbusMaster.h>

//#define MAX485_RE_NEG  2
//#define MAX485_DE      3
#define ESPLED1          25
#define ESPLED2          26
#define ESPLED3          27
#define DEnRE485  14
#define RXPIN     12
#define TXPIN     2
#define BAUD      9600
#define SLAVE_ID 1

ModbusMaster node;

void preTransmission()
{
  digitalWrite(DEnRE485, 1);
//  digitalWrite(MAX485_RE_NEG, 1);
//  digitalWrite(MAX485_DE, 1);
}


void postTransmission()
{
  digitalWrite(DEnRE485, 0);
//  digitalWrite(MAX485_RE_NEG, 0);
//  digitalWrite(MAX485_DE, 0);
}

void setup() {
  // Initialize control pins
  pinMode(ESPLED1, OUTPUT);
  pinMode(ESPLED2, OUTPUT);
  pinMode(ESPLED3, OUTPUT);
  pinMode(DEnRE485, OUTPUT);
  digitalWrite(DEnRE485, 0);

  // Modbus communication runs at 9600 baud
  Serial.begin(115200);
  Serial1.begin(BAUD, SERIAL_8N1, RXPIN, TXPIN);
  delay(1000);
  Serial.println("Esp32 started");

  // Modbus slave ID 1
```



SHT Modbus module to PD.Charger_GSM v.2.5 connection diagram


```
  node.begin(SLAVE_ID, Serial1);

  // Callbacks allow us to configure the RS485 transceiver correctly
  node.preTransmission(preTransmission);
  node.postTransmission(postTransmission);
}
int16_t cnt;

void loop() {
  // Request 2 registers starting at 0x0001
  uint8_t result = node.readInputRegisters(0x0001, 2);
  Serial.print("Data Requested..");

  if (result == node.ku8MBSuccess) {
    // Get response data from sensor
    Serial.print("Temperature: ");
    Serial.print(float(node.getResponseBuffer(0) / 10.00F));
    Serial.print("   Humidity: ");
    Serial.println(float(node.getResponseBuffer(1) / 10.00F));
  }
  delay(1000);
  if (++cnt & 1)
    digitalWrite(ESPLED3, 1);
  else
    digitalWrite(ESPLED3, 0);}
```

