The project is energy management, controls and monitoring system powered by esp32 chip . the system is fited in an energy pack that contains hybrid inverter and battery (lithium 48v pack voltage) with integrated bms which has rs385 and canbus interface. The system is supposed to monitor input energy from the grid and energy output of the inverter (home consumption) and also monitor the battery bms to fetch soc and battery general activities for mobile app and other oprations , also include automations and modes options , controls through the relays and then feeds the data to orange ((mqtt local broker)pi zero) through canbus to usb so that pi zero can locally broadcast to mobile app monitor and also let the pi zero bride the same data to web mqtt server for controls and monitor when mobile app is away from home wifi

Hardware component details:

1: Esp32  
2 : 4 50A AC current sensors (with PCB connectors)  
3: 4 relays 16A (PCB mounted )  
4: canbus,rs485 and UART  
4: RGB led WS2812B(pcb connector)  
5: 7 segment display TM1637 (PCB connector)  
6: a small separate board with 3 led indicators (with a connector to the main board)  
7: 24v power system   
8: board size 80\*180mm  
  
(Make all the remaining unused esp32 pins available on the pcb and labeled)  
...............  
IoT energy monitoring system:  
  
Microcontroller  
  
1. ESP32-WROOM-32-N4   
  
  
  
Current Sensors  
  
2. 2 CT external Sensors (50A) for Grid and Home monitoring  
  
  
3. 2 CT external Sensors (50A) for External Loads  
  
  
4. Burden Resistors (68Ω, 0.5–1W)  
  
  
5. 3.3V Zener Diodes  
  
  
6. Low-pass Filters (1kΩ resistor + 100nF capacitor)  
  
  
  
Relays & Control  
  
4 Relays (16A, SPDT/SPST) 1 controls grid in and another controls ac out from the inverter (house consumption, the remaining 2 are for external loads controls   
  
  
Optocouplers (PC817)  
  
  
 Flyback Diodes (1N4007)  
  
  
  
Communication Interfaces  
  
RS485 Half-Duplex  
  
IC: MAX485  
  
120Ω Termination Resistor  
  
TVS Diodes (SMAJ33A)  
  
  
CAN Bus Half-Duplex  
  
IC: SN65HVD230  
  
120Ω Termination Resistor  
  
TVS Diodes (SMAJ33A)  
  
  
UART Full-Duplex  
  
UART1: TX (GPIO 26), RX (GPIO 27)  
  
TVS Diodes (SMAJ33A)  
  
  
  
User Interface Components  
  
Status LEDs  
  
Green LED: GPIO 2  
  
Red LED: GPIO 15  
  
Yellow LED: GPIO 13  
  
Resistors: 220Ω each  
  
  
RGB LED  
  
WS2812B RGB LED: GPIO 25   
  
100nF Decoupling Capacitor  
  
  
7-Segment Display  
  
TM1637 (CLK: GPIO 14, DIO: GPIO 12)  
  
4.7kΩ Pull-up resistors (CLK, DIO lines)  
  
  
  
Connectors & Terminals  
  
CT Connectors:  
  
4 × Phoenix Contact MSTB 2.5/3-ST-5.08 terminal blocks (3-pin)  
  
  
RS485 & CAN Connectors:  
  
2 × Phoenix Contact MC 1.5/4-ST-3.81 screw terminals (4-pin)  
  
  
UART Connector:  
  
Phoenix Contact MC 1.5/4-ST-3.81 or equivalent (4-pin)  
  
  
Relay Connectors:  
  
4 × High-current PCB terminal blocks (≥16A, 5.08mm pitch)  
  
  
Power Connector:  
  
Terminal Block (24V DC, 5A-rated)  
  
  
  
Power Supply & Regulation  
  
24V DC input  
  
Fuse (5A)  
  
DC/DC Buck Converter (LM2596, 24V→5V)  
  
LDO regulator (AMS1117-3.3, 5V→3.3V)  
  
Decoupling Capacitors (100µF, 10µF, 100nF)  
  
  
PCB & Miscellaneous  
  
FR4 PCB, 4-layer, ENIG finish, 1.6mm thickness  
  
Silkscreen labels clearly identifying connectors, GPIOs, and test points  
  
Mounting holes (4 × M3)  
  
.,.............................  
  
Here's a summarized GPIO table clearly defining all ESP32 pin assignments for your PCB:  
  
Notes:  
  
ADC pins are carefully selected to avoid conflicts with digital functionality.  
  
Relays are driven via isolation components (optocouplers or transistor circuits).  
  
RS485 uses standard half-duplex communication with a MAX485 transceiver.  
  
CAN Bus is implemented with a dedicated transceiver SN65HVD230.  
  
Full-duplex UART reserved for future expansions or external communications.  
  
WS2812B requires a single GPIO for control, simplifying LED management.  
  
  
This GPIO summary provides a concise reference for your PCB designer.

]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

Firmware core and function

Firmware structure and identifier  
Firmware name : Alfaspark  
Device software version: 2.5  
Device hardware no :....(The esp32 serial number )  
Device type : Energy-Pack(for Energy-pack) for this project or ((Energy-gateway) for the gateway app project)

]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

**Checklist for Ensuring Firmware Stability and Data Integrity**

**1. Memory Management**

* Avoid memory leaks (tools: Valgrind or ESP32 heap diagnostics).
* Protect against stack overflow (allocate sufficient stack space).
* Monitor dynamic memory allocation.

**2. Non-Volatile Storage (NVS) Integrity**

* Limit write frequency to NVS to avoid exceeding write cycles.
* Implement wear-leveling for flash memory.
* Use write-throttle mechanisms for persistent data storage.

**3. Task Scheduling and Interrupt Management**

* Synchronize tasks using mutexes, semaphores, or event groups.
* Avoid overloading the CPU; optimize task priorities and timing.
* Manage interrupt priorities and avoid blocking critical tasks.

**4. Robust Error Handling**

* Detect and handle communication errors using CRC or checksums.
* Recover gracefully from errors to avoid device lock-ups.
* Use exception handling mechanisms and log faults.

**5. Watchdog Timer**

* Enable hardware and software watchdog timers to reset the device if it becomes unresponsive.
* Monitor critical task execution with software watchdogs.

**6. Data Integrity**

* Validate data with checksums or range checks before use.
* Store critical data in multiple locations as backups.
* Add mechanisms for detecting and correcting corrupted data.

**7. Communication Protocols (RS-48/5CANBUS)**

* Implement retry mechanisms for failed communications.
* Use timeouts to avoid indefinite blocking in case of errors.
* Include acknowledgment protocols for master-slave communication.

**8. Firmware Updates**

* Implement OTA updates with fallback mechanisms (e.g., dual-bank firmware).
* Track firmware versioning for compatibility.
* Validate new firmware before applying.

]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

MODES functions  
(Self consumption )  
when on, open(disconnect) the grid in relay when the battery soc is above 60% and closes(connect) the relay when battery soc is below 20%.  
  
  
(Time of use)  
When turned on user selects the time based on am and pm( eg tou 8am -4pm or pm - am or am - am or pm - pm) the inverter output(home consumption) relay open(disconnect) for none time of use and closes(connects) for time of use.  
  
(backup)  
When on the grid relay will permanently stays closed(connected)

(grid daily limit)

When this is selected user input capacity daily cap in kw, the system calculates grid input kw through ct and when the limit kw is reached it opens(disconnects) the grid input relay and override any other function or command to grid relay until grid daily limit is turned off, system resets for new cycle of grid daily limit by 12 midnight

Modes potential conflict : (grid daily limit) can work with other modes but its grid relay control is on top control it overrides all other grid relay controls, eg (backup) mode and (time of use mode gird relay controls are overridden by (grid daily limit)  
 when the modes get to the point of overridden grid relay they fall back to functioning with what ever the grid relay state is.  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

Energy parameter calculation

Energy calculation is in kw and its in one decimal place like 1.2kw for 1.2kw or 0.990kw for 990w…

The current sensor calculate capacity based on current multiply by 220v(by default)= the capacity in kw.. goes for the 4 ct current sensors

Battery capacity parameters is based on the bms fetched data . to calculate the battery discharge is going to be battery pack voltage multiply by battery charge or discharging current… like to calculate the battery’s charging capacity its going to be battery pack voltage x current interring the battery pack = kw

]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

Battery soc mapping…(Note! Battery soc is not base on bms original soc rather it is based on alfaspack soc mapping) alfaspark connected to the bms rs485 to fetch battery voltage ……. Here's a simplified and clear summary tailored exactly for your ESP32 firmware:

Here's a concise summary based on your specified voltage range:

**Battery State-of-Charge (SoC) Voltage Summary for 16s LFP (48 V) Pack:**

| **Condition** | **Per Cell Voltage** | **Pack Voltage (16s)** | **Reported SoC** |
| --- | --- | --- | --- |
| **Fully Discharged (0%)** | ≤ 2.90 V | ≤ 46.40 V | **0%** |
| **Fully Charged (100%)** | ≥ 3.37 V | ≥ 53.92 V | **100%** |
| **Intermediate Range (Linear)** | 2.90–3.37 V | 46.40–53.92 V | Linear (0–100%) |

**Simple Firmware Formula (for ESP32):**

cpp

CopyEdit

float calculateSoC(float packVoltage){

if (packVoltage <= 46.4)

return 0.0;

else if (packVoltage >= 53.92)

return 100.0;

else

return (packVoltage - 46.4) \* 13.3;

// (100 / (53.92 - 46.4)) ≈ 13.3% per volt

}

**Explanation of Formula:**

* Voltage Span: **53.92 V − 46.40 V = 7.52 V**
* SoC per volt increment: **100% ÷ 7.52 ≈ 13.3% per volt**

This simplified and efficient approach ensures accurate SoC calculations within your specified voltage range.

[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[

SOLAR PRODUCTION

The system design don’t originally have a dc current/voltage sensor integrated for tracking solar input so its going calculated based on home consumption battery charging or discharge capacity and then subtracted by the gird input = the solar production in kw the formular goes like

Fomular

To calculate \*\*solar production\*\* in your Alfaspark-based system without a dedicated solar sensor, use the energy conservation principles and the data you already collect: \*\*grid input\*\*, \*\*home consumption\*\*, and \*\*battery charge/discharge power\*\*. Here's the step-by-step solution:

---

### \*\*Key Formulas\*\*

1. \*\*When Battery is Charging\*\*:

\[

\text{Solar Production} = \text{Home Consumption} + \text{Battery Charge Power} - \text{Grid Input}

\]

\*Why?\* Solar powers the house \*\*and\*\* charges the battery. Any grid input reduces the solar contribution.

2. \*\*When Battery is Discharging\*\*:

\[

\text{Solar Production} = \text{Home Consumption} - \text{Grid Input} - \text{Battery Discharge Power}

\]

\*Why?\* Solar and battery together power the house. Grid input only supplements if solar + battery are insufficient.

3. \*\*When Battery is Idle\*\*:

\[

\text{Solar Production} = \text{Home Consumption} - \text{Grid Input}

\]

\*Why?\* Solar directly powers the house, and grid fills the gap.

---

### \*\*Implementation Steps\*\*

#### 1. \*\*Measure Power Values\*\*:

- \*\*Grid Input (kW)\*\*:

\[

\text{Grid Input} = \text{Grid Current (A)} \times 220V / 1000

\]

- \*\*Home Consumption (kW)\*\*:

\[

\text{Home Consumption} = \text{Inverter Output Current (A)} \times 220V / 1000

\]

- \*\*Battery Power (kW)\*\*:

- \*\*Charging\*\*: `Battery Current > 0`

\[

\text{Battery Charge Power} = \text{Battery Current (A)} \times \text{Battery Voltage (V)} / 1000

\]

- \*\*Discharging\*\*: `Battery Current < 0`

\[

\text{Battery Discharge Power} = |\text{Battery Current (A)}| \times \text{Battery Voltage (V)} / 1000

\]

#### 2. \*\*Determine Battery State\*\*:

- Use BMS data to check the sign of battery current:

- \*\*Positive Current\*\*: Battery is \*\*charging\*\*.

- \*\*Negative Current\*\*: Battery is \*\*discharging\*\*.

- \*\*Near Zero Current\*\*: Battery is \*\*idle\*\* (use a small threshold, e.g., ±0.1A, to avoid noise).

#### 3. \*\*Calculate Solar Production\*\*:

- \*\*Example Code (C++ Pseudocode)\*\*:

```cpp

float grid\_input = grid\_current \* 220.0 / 1000.0; // kW

float home\_consumption = inverter\_current \* 220.0 / 1000.0;

float battery\_power = battery\_current \* battery\_voltage / 1000.0; // kW

float solar = 0.0;

if (battery\_current > 0.1) { // Charging

solar = home\_consumption + battery\_power - grid\_input;

} else if (battery\_current < -0.1) { // Discharging

solar = home\_consumption - grid\_input - abs(battery\_power);

} else { // Idle

solar = home\_consumption - grid\_input;

}

if (solar < 0) solar = 0; // Solar cannot be negative

```

---

### \*\*Edge Cases & Validation\*\*

1. \*\*Grid Charges Battery\*\*:

- If the grid charges the battery (e.g., home uses 2kW, grid provides 3kW, battery charges 1kW):

\[

\text{Solar} = 2 + 1 - 3 = 0 \ \text{kW (correct, grid handles all)}.

\]

2. \*\*Solar + Battery Power Home\*\*:

- If solar = 3kW, battery discharges 1kW, home uses 4kW, grid = 0:

\[

\text{Solar} = 4 - 0 - 1 = 3 \ \text{kW (correct)}.

\]

3. \*\*Negative Solar (Error Handling)\*\*:

- If the calculation yields `solar < 0`, set it to `0` (invalid instrumentation or edge case).

---

### \*\*Firmware Workflow\*\*

1. \*\*Data Collection\*\*:

- Sample grid/inverter currents and battery voltage/current at fixed intervals (e.g., every 1 second).

2. \*\*Power Calculation\*\*:

- Convert currents to power values (kW).

3. \*\*State Detection\*\*:

- Use battery current direction to determine charge/discharge state.

4. \*\*Solar Calculation\*\*:

- Apply the appropriate formula based on battery state.

5. \*\*Display\*\*:

- Show real-time solar power (kW) and accumulated solar energy (kWh) in the app.

---

### \*\*Visualization\*\*

| \*\*Scenario\*\* | Battery State | Formula |

|-----------------------------|----------------|----------------------------------|

| Solar powers home + battery | Charging | `Solar = Home + Charge - Grid` |

| Solar + battery power home | Discharging | `Solar = Home - Grid - Discharge` |

| Solar powers home directly | Idle | `Solar = Home - Grid` |

---

This method ensures accurate solar calculations using existing sensors and aligns with Alfaspark’s firmware capabilities.

OPTION 2 /\*

===========================

ALFASPARK: Solar production Solarize with 2 external loads sensors

===========================

System Overview:

- ESP32-based real-time energy monitor

- 4 AC current sensors:

[1] Grid Input

[2] Inverter Output (Home Consumption)

[3] external load 1

[4] external load 2

- RS485: Fetch battery voltage and current from BMS

- CAN Bus: Internal firmware communication

- 4 relays:

[1] Grid Disconnect

[2] Inverter Output Disconnect

[3] Pool Pump Control

[4] Water Heater Control

Core Logic: Real-Time Solar Production Estimation (every 2 seconds)

--------------------------------------------------------------------

Battery current sign indicates state:

- Positive (+): Charging

- Negative (−): Discharging

- Zero (≈0): Idle

Formulas:

----------

If battery is \*\*charging\*\*:

Solar = (Home + Battery) − Grid

If battery is \*\*discharging\*\*:

Solar = Home − Grid − abs(Battery)

If battery is \*\*idle\*\*:

Solar = Home − Grid

Clamp:

If Solar < 0 → Solar = 0

External loads Monitoring:

- 1 = Sensor 3 × 220V

- 2 = Sensor 4 × 220V

- add this to home consumption too

(solar production = home+external load ….)

Output:

- All readings updated every 2s

]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]] alfaspark to bms communication

Communicates through rs485 and bms datasheet will be provided for this

]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

Alfaspark to alfasparks canbus communication

Its based on 3 alfaspark connected through there canbus to talk to each other some kind of master and slave has to be applied and the alfasparks will elect for the master based on ids of their hardware also an identifier of device type=(energy-pack)means they are all the same system also each and every one of the alfaspark will have its own unique Identity after the election and re election will happen when new alfaspark is introduced or if master isn’t active for 2 minutes

]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

alfaspark to orange pi zero

communication through afterspark canbus to pi zero usb , able to supply pi zero with all the data for mobile app monitor and also feedback controls to alfaspark through the mobile app.. e.g modes selections….