EME Rotator Controller

Architecture Dual-Core ESP32-S3 ProS3 — Station ON7KGK — v7.4

1. Vue physique — Où va quoi

Le système est réparti sur **trois emplacements physiques** reliés par des câbles dédiés.

BOITIER ÉTANCHE AU PIED DU MÂT (rotateur)	
• **ESP32-S3 ProS3**	Le cerveau principal — tout passe par lui
• **MC33926 dual driver**	Contrôle les 2 moteurs AZ + EL (24V)
• **MCP23017 (I2C)**	4 fins de course + 4 boutons + directions moteur + LEDs
• **W5500 Ethernet (SPI3)**	Lien réseau vers le shack (CAT6, 50-100m)
• **AS5048A encodeur AZ (SPI)**	Magnétique 14-bit, monté sur slewing drive AZ
• **Hall 12 PPR AZ (PCNT)**	Encodeur incrémental sur moteur AZ
• **GPS NEO-6M (UART2)**	Heure UTC + PPS, antenne sur le toit du boîtier
• **OLED 128×64 (I2C)**	Affichage local position/état
• **EEPROM/FRAM (I2C)**	Sauvegarde position (double buffering + CRC)
• **Bouton STOP (GPIO7)**	Sécurité hardware (coupe EN du MC33926)
• **MAX485 RS-485 (UART1)**	Vers le bras parabole (câble 2-3m)

■ **Câble RS-485 paire torsadée (2-3m)**

▼

BRAS DE LA PARABOLE (avec équipement 10 GHz)	
• **Arduino Nano R4 (Modbus addr 1)**	Télémétrie : tensions 24V/12V, puissance PA, 3× températures, relais anti-gel
• **HWT901B inclinometre (Modbus addr 2)**	Élévation vraie de la parabole (gravité, 0.05°) + gyroscope pour PID
• **Transverter 10 GHz + PA + LNA**	Équipement RF (non contrôlé par l'ESP32)

■ **Câble Ethernet CAT6 (50-100m)**

▼

SHACK (intérieur)	
• **Alimentation 24V > 5A**	Alimente moteurs + ESP32 + Nano R4
• **Switch Ethernet**	Relie ESP32 au réseau local
• **NUC i5**	WSJT-X + PSTRotator (ou app custom) + LimeSDR

2. Récapitulatif des connexions

Composant	Connecté à	Où physiquement
MCP23017	ESP32 (I2C)	Boîtier rotateur
GPS NEO-6M	ESP32 (UART2)	Boîtier rotateur
OLED 128×64	ESP32 (I2C)	Boîtier rotateur
EEPROM/FRAM	ESP32 (I2C)	Boîtier rotateur
W5500 Ethernet	ESP32 (SPI3)	Boîtier rotateur
AS5048A enc. AZ	ESP32 (SPI bit-bang)	Sur slewing drive AZ
Hall AZ	ESP32 (PCNT)	Sur moteur AZ
MC33926 drivers	ESP32 (MCPWM+MCP)	Boîtier rotateur
Bouton STOP	ESP32 (GPIO7) + MC33926	Boîtier rotateur
4 fins de course	MCP23017 (via optos)	Structure rotateur
4 boutons manuels	MCP23017	Boîtier rotateur
Nano R4	ESP32 (RS-485, addr 1)	Bras parabole (2-3m)
HWT901B élévation	ESP32 (RS-485, addr 2)	Bras parabole (2-3m)

Tout passe par l'ESP32. Le Nano R4 est un esclave de télémétrie au bout d'un câble RS-485 (2-3m).

3. Core 1 — Temps réel (critique)

Le Core 1 est dédié exclusivement au **contrôle moteur et aux capteurs**. Pas de réseau, pas d'affichage, pas de parsing. La boucle PID à 100 Hz doit être **garantie** — si elle rate un cycle, le moteur peut osciller ou dépasser la cible.

Tâche	Priorité	Période	Fonction
`task_pid_loop`	MAX-1	10 ms	PID + PWM moteurs + ADC courant + direction MCP23017 + anomalies
`task_enc_read`	MAX-2	1000 ms	Lecture AS5048A (ou HH-12) + recalib PCNT + sauvegarde EEPROM
`ISR stop_btn`	ISR	Event	Bouton STOP (GPIO7)
`ISR mcp_int`	ISR	Event	MCP23017 interrupt (fins de course, boutons, SF)
`ISR gps_pps`	ISR	Event	GPS PPS (1 pulse/seconde)

4. Core 0 — Communication (tolérant à la latence)

Le Core 0 gère tout ce qui est **communication**. Si l'OLED met 5 ms de plus à se rafraîchir ou si un paquet TCP arrive avec 10 ms de retard, aucune conséquence. Les priorités assurent que TCP (PSTRotator / app) est traité avant l'OLED.

Tâche	Priorité	Période	Fonction
`task_tcp_gs232`	5	Event	Serveur TCP PSTRotator (port 4533)
`task_tcp_app`	5	Event	Serveur TCP app Windows JSON (port 4534)
`task_modbus`	4	1000 ms	Polling RS-485 : Nano R4 (addr 1) + HWT901B (addr 2)
`task_gps`	4	100 ms	Parsing NMEA GPS (UART2)
`task_oled`	2	500 ms	Mise à jour affichage OLED

5. Flux de données entre les deux cores

Les données partagées entre cores passent par des **variables atomiques** ou des **queues FreeRTOS** — pas de mutex lourd qui bloquerait le PID.

Donnée	Direction	Écrit par	Lu par	Mécanisme
`target_az, target_el`	Core 0 → Core 1	TCP (GS-232 / App)	PID loop	atomic float
`current_az, current_el`	Core 1 → Core 0	PID loop	TCP servers	atomic float
`state (machine état)`	Core 1 → Core 0	PID loop	TCP + OLED	atomic uint8
`hwt901b_el, gyro`	Core 0 → Core 1	Modbus task	PID loop (EL)	atomic float
`nano_telemetry`	Core 0 → Core 0	Modbus task	TCP app	struct copie
`utc_time`	Core 0 → tous	GPS task + PPS ISR	Solar cal, OLED	atomic uint64

6. Principe fondamental

> **Core 1 ne bloque JAMAIS.**
>
> Il lit et écrit des valeurs atomiques, c'est tout. Core 0 peut bloquer sur TCP accept, Modbus timeout, etc. — ça n'affecte pas le PID.
>
> C'est le gros avantage du dual-core par rapport à un Arduino classique : le réseau ne peut pas perturber le contrôle moteur, même si un client TCP se déconnecte brutalement ou si le Modbus timeout.

7. Diagramme de communication inter-cores