FIRMWARE MANUAL

DlaBoard_B

DlaBoard_B · 出厂固件 · 使用说明书（含 CAN 协议全文）

副标题	DlaBoard_B · 出厂固件 · 使用说明书（含 CAN 协议全文）
品牌	Daliang Auto
产品型号	DlaBoard_B（硬件已定版，原理图 2026-08-01）
固件版本	**0.1.13**（构建标识 `6d7c7d9`，Release 构建，2026-09-11）
镜像校验	62000 字节，CRC32 `9fae03e9`（镜像自述头已盖章）
文档版本	Rev. 1.6
发布日期	2026-09-11
控制台	USB1（USART1），115200 8N1，CR/LF
整车总线	经典 CAN，1 Mbps，标准帧，DLC = 8

本书与规格书的分工

- **《DlaBoard_B 产品规格书》描述硬件**：电气参数、引脚、连接器、I/O 资源、机械尺寸。 硬件不改版，规格书就不改。
- **本书描述出厂固件**：整车控制逻辑、CAN 协议、串口协议、诊断命令、参数、升级方式。 **固件每发布一版，本书跟着走一版。**
- 两本书都出现的量（供电范围、引脚、连接器针序），**以规格书为准。**

⚠ **本书把「已验证」和「未验证」分开写，请不要合并阅读。** 这块板处在整车集成阶段， 有一批功能代码已经实现、但还没有在真车上验证过。哪些验过、哪些没验过，见 §13。 一份把两者混在一起的说明书，会让人把没验过的东西当成保证。

⚠ **本板不含功率级。** 它不驱动电机，也没有栅极驱动。它的输出是**总线上的请求**， 由电机驱动器执行。它的「急停」是软件的 —— 见 §11 安全申报。

---

1. 这套固件是什么

DlaBoard_B 是整车里**司机意图的归属方**：它读踏板、档位、整车使能、刹车开关、遥控和 车载计算机，把这些互相矛盾的输入仲裁成**一份行驶请求**，每 20 ms 通过 CAN 发给电机驱动器。

它也是整车里**唯一能回答「这台车为什么不走」的地方** —— 不允许驱动的理由有十四条， 从外面看现象完全一样（车不动），固件把它们逐条分开报（§7）。

1.1 四条对外通路

通路	对面是谁	物理口	章节
整车 CAN	电机驱动器（一块双电机板，或两块单电机板）	CN34 / CN32	**§5（专章）**
车载计算机串口	Jetson 等自动驾驶计算机	**USB2**（UART5）	§6
控制台 / 上位机串口	人，或 PC 上位机软件	**USB1**（USART1）	§2 – §3
触摸屏	车上的仪表与按键	**CN35**（USART2，9600）	§4

板上每个连接器出厂固件拿来做什么，见 §1.3。 那一节还列出了固件不使用的口。

两个 USB 口不等价，也不可互换。 USB1 上只跑 ASCII 文本，USB2 上只跑与车载计算机的 二进制帧。这两根线在 0.1.0 之前是同一根，代价是自动驾驶模式下文本命令全部问不出来； 分线之后，**任何模式下控制台都能用。**

1.2 六种模式

模式	谁在开车
`MANUAL`	人：踏板 + 档位 + 整车使能
`REMOTE`	遥控器
`TRAINING`	示教录制
`COLLECT`	采集（SLAM）：行为同示教，另上报采集帧与 IMU
`FOLLOW`	车载计算机跟随（`0x10` 指令帧）
`AUTO`	车载计算机自动驾驶（`0x11` 指令帧 + 速度闭环，见 §6）

模式由触摸屏按键切换。**无论哪种模式，§7 的十四条阻止理由一条都不放松。**

1.3 接口功能定义：出厂固件把每个口用来做什么

《产品规格书》回答的是「这个连接器接到哪个 MCU 引脚」，本节回答两件事： **出厂固件拿它做什么**，以及**这个口收发的是什么信号、什么格式。**

针序、间距、板载分压、电平耐压一律看规格书（§7 通信接口、§8 通用 I/O、§11 连接器一览表）， 本节不重复引脚映射。

> 换一版固件，本节就可能变；规格书那几节不会变。这正是两本书分开的原因。

1.3.1 一眼看完：哪个口做什么

连接器	MCU 资源	出厂固件的用途	收发什么
USB1	USART1	控制台 / 上位机	ASCII 文本行，115200 8N1，CR/LF。见 §1.3.2
USB2	UART5	车载计算机	22 字节定长二进制帧，115200 8N1。见 §6
CN34 / CN32	CAN1（PB9 / PB8）	整车总线	经典 CAN 1 Mbps，标准帧，DLC = 8。见 §5
CN35	USART2（PD5 / PD6）	**触摸屏**	DGUS / VGUS 变量帧，**9600 8N1**。见 §1.3.2
U7（5P）	PD0 / PD1 / PD3 / PD4	**PRND 档位开关**	四路干接点，低有效。脚 1 = 公共端接 GND，脚 2 = P、3 = R、4 = N、5 = D，见 §4.2
CN22	INPUT_PD10	**整车使能开关**	一路干接点，低有效。断开 = 不允许驱动，同时是一路刹车意图
CN5 / CN6	PE2 / PE3	**刹车开关，两路**	干接点，低有效。两路是「或」，任一路踩下即刹车意图（见 §1.3.4 第一条）
CN30	ADC1_IN8 / PB0	**油门踏板**	**0 – 5 V 直流模拟量**，见 §1.3.2
H2 · RC1	PC8 / TIM3_CH3	遥控转向	舵机 PWM。本板解析并回报，但转向字段恒发 0，见 §5.10
H2 · RC2	PC9 / TIM3_CH4	遥控油门	舵机 PWM。`REMOTE` / `TRAINING` / `COLLECT` 下有效
H2 · RC3	PC7 / **USART6_RX**	**iBUS 串口遥控**（默认）	六路里唯一落在 USART 接收脚上的。`rc_source=0` 时这一路用不了
H2 · RC4	PC6 / TIM3_CH1	**远程急停**（`rc_source=0` 的 PWM 模式）	舵机 PWM 当两位开关用，> 1500 µs = 刹车。⚠ 0.1.4 以前这一路在 RC3
H1	SWD	烧录 / 在线调试	见规格书 §10.1

板载器件不是对外连接器，但固件用到，一并列在这里：

器件	接口	出厂固件的用途
6 轴 IMU	SPI1	`COLLECT` 模式的采集帧与 `IMU?`。初始化失败不阻塞主流程，IMU 字段填 0
EEPROM 8 KB	I2C1	参数持久化，双区轮换 + CRC32，见 §8
输入电压检测	ADC1_IN11 / PC1	`tlm` 里的板供电压
用户 LED	PE0	**上电点亮后不再变化**，不用它表示状态
蜂鸣器	PE1	出厂固件运行时不鸣
用户按键	PC13	出厂固件不读

1.3.2 每一路收的是什么信号

油门踏板（CN30）：0 – 5 V 直流模拟量

项目	内容
接口收什么	**0 – 5 V 直流电压**。踏板里的电位器或霍尔式踏板都行，只要输出是这个量程
连接器	2.0 mm 3P：**1 = GND / 2 = 信号 / 3 = +5 V**（取自板上 5 V 轨，可直接给踏板供电）

项目	内容
板上前端	10 kΩ 串阻 + 20 kΩ 对地 + 100 nF ⇒ 5 V 输入落到 ADC 上约 3.33 V
固件怎么读	12 位 ADC（0 – 4095），每趟主循环取一次，按两个端点线性映射成 **0 – 1000‰**
两个端点	参数 `thr_adc_dead`（松开点，默认 850）、`thr_adc_full`（踩到底，默认 3500），单位是 ADC 计数。`calib start pedal` 就是把这两个数量出来，见 §9
回位间隙	参数 `thr_dead`（默认 30‰）：小于它的一律当 0
读数低于松开点，或两个端点被设反	**这一脚当没踩**（0‰），不夹一个假比例出来
怎么确认接对了	`tlm` 的 `veh` 行里有踏板‰，踩一脚看它动；`selftest run` 的 `pedal_adc` 报的是原始 ADC 计数

> ⚠ **踏板线掉了不会报错。** 信号脚一断，板上那只 20 kΩ 把 ADC 拉到 0 V，读出来就是「松开」。 方向是安全的（车不会自己加速），但**没有任何东西会说线断了** —— 现象和司机没踩脚一模一样。

档位、整车使能、刹车（U7 / CN22 / CN5 / CN6）：干接点

项目	内容
接口收什么	**无源触点** —— 机械开关、继电器触点、行程开关。合上 = 接到 GND = 有效
有效电平	**低有效**。板上串 100 Ω，**MCU 内部上拉已开**，不接线的脚稳定读高
⚠ 不要从外部灌电压	这几路按干接点设计，接一个开关就够。GPIO 是 3.3 V 的，电平耐压见规格书 §13
触点要求	档位那四路**必须是四位置选择开关**，当前档位那一路持续闭合，见 §4.2。刹车与使能是普通开关
去抖	档位：解出来的档位要稳定 **50 ms** 才生效，开机 **500 ms** 内不解码；**使能与刹车没有软件去抖**，跟着触点走
断线会怎样	档位：四路全高 ⇒ 解不出 ⇒ **落 P**。使能：读高 ⇒ 不允许驱动。刹车：读高 ⇒ 没有刹车意图 ⚠ 这一路断了不会被发现

遥控：两种模式，二选一

⚠⚠ **iBUS 和 PWM 不是两套可以并用的通道，是两种模式。** 物理上就不允许并存：走 iBUS 时 RC3 那个引脚（PC7）是串口，那一路脉宽根本 不存在；而 iBUS 一根线就把十几路都送来了，其余五个脚没意义。

用参数 `rc_source` 选：`0` = PWM，`1` = iBUS。**出厂默认 iBUS。** 现在听哪一种，`RC?` 里直接报 `src=ibus` 或 `src=pwm` —— 别靠猜。

iBUS 模式（默认）

项目	内容
接哪里	**H2 的 RC3**（信号排从 RC6 那头数第 4 个）。六路里只有它落在 USART 接收脚上
协议	FlySky iBUS，115200 8N1 不反相，32 字节一帧，约 7 ms。通道值 1000 – 2000，和舵机脉宽同量纲
⚠ SBUS	**不支持。** SBUS 是反相信号，而 STM32F4 的 USART 没有 RX 反相位 —— 要用得外接一个反相器
通道分配	ch1 转向、1ch2 油门、ch5 油门限速、ch7 远程急停、ch6 转向限速
坐标参考	富斯 i6X 开 10 通道时：ch1/ch2 右摇杆，ch5/ch6 = VrA/VrB 旋钮，ch7 = SwA 二位开关
坏帧	校验和不对、或通道值越界 ⇒ **整帧丢**，不拿半截。`RC?` 里 `err=` 数得出来
怎么看	`RC?` 报前八路原始值 + 帧数 + 坏帧数 + 三态

⚠⚠ 油门为什么是 ch2 而不是 ch3

航模遥控器的 ch3（左摇杆上下）是**带阻尼不回中**的油门杆。对车来说那是 两件坏事：松手不减速；而且上电那一刻杆在哪儿都有可能，一使能车就窜出去。 ch2（右摇杆上下）自动回中，中位 = 停。

两个限速旋钮，分开管两件事

旋钮	管什么	⚠ "不限"是哪个值
ch5 (VrA)	驱动油门上限	**1000 = 不限**；0 就是真的不走
ch6 (VrB)	转向速度上限	**0 = 用转向板自己的默认上限**（即最快），因此旋钮拧到底会夹到 1

⚠ 两边的"不限"是不同的数、而且反着 —— 转向那个 0 的含义由 CAN 契约定死。 搞反的后果是一边车不动、另一边方向机全速。

分开的理由：合成一个旋钮的话，想把车调慢就得连转向一起调慢 —— 而低速挪车恰恰是最需要方向机跟得上的时候。

PWM 模式（`rc_source = 0`）

项目	内容
接口收什么	常规接收机的 **PWM 脉冲**，约 50 Hz（20 ms 一帧）
脉宽	1000 – 2000 µs，**中位 1500 µs**。固件量的是**正脉宽**（上升沿到下降沿）
每一路的接法	H2 是 3 × 6 排针，每个通道一组 信号 / +5 V / GND，接收机的舵机线直接插
用哪几路	RC1 转向、RC2 油门、**RC4 远程急停**。⚠ 没有限速旋钮
⚠ RC3	**用不了** —— 那个引脚已经给了串口。急停因此从 RC3 搬到了 RC4
急停判据	**> 1500 µs = 刹车**。要用**两位置开关**通道，它自己保持位置；用弹回的按钮会一松手就松刹车

两种模式共用的部分

项目	内容
失效判据	**200 ms** 没有新帧就算失效（参数 `rc_stale`，50 – 2000 ms 可改）
三态	`NEVER` 从来没接过 ⇒ 不主张任何东西； `UP` 在线； `LOST` 曾活过现在断了 ⇒ 摇杆交中位， `REMOTE` / `COLLECT` 下强制刹车。见 §4.4
⚠ 不活的那一种	**一句话不说，包括不主张刹车。** 否则插着 iBUS 的车会因为"那根没插的 PWM 线断了"而永远刹着

⭐ **iBUS 比 PWM 更需要那条三态判据。** PWM 断线时信号线被下拉拉低、捕获自然停； 而 iBUS 断线时**最后一帧的通道值还好端端地留在内存里**，光看通道值和 "遥控器还在打"一模一样。只有帧计数和时间戳说得出真相。

⭐ **2026-09-09 实测（富斯 i6X + 配套接收机）**：遥控器关机后这台接收机 **就不发了**，不是继续发失控保护值 ⇒ 200 ms 后判 LOST、强制刹车。 ⚠ **换接收机要重新量**：有的型号会继续发，那时链路在板子看来是健康的而人已经不在， 只能靠在接收机里把急停通道的失控值设成刹车位。

触摸屏（CN35）：DGUS / VGUS 变量帧

项目	内容
串口参数	**9600 8N1** —— 和两个 USB 口的 115200 不是一个波特率
帧格式	`A5 5A <LEN> <CMD> <ADDR_HI> <ADDR_LO> <数据…>` 。 `LEN` 从 `CMD` 那一字节起算，多字节量**大端**，**没有校验和**

项目	内容
板 → 屏	`CMD = 0x82` 写变量，一次一个 16 位字。文本控件写 UTF-8 字节并以 `FF FF` 收尾（用来清掉控件里的残留字符）
屏 → 板	`CMD = 0x83` 变量回读：屏把地址与键值报上来，固件取第一个字当键值
屏发给板的	模式 `0x05A1`、档位 `0x05E1`、速度档 `0x05C1`
板发给屏的	车速 `0x02E5`、连接状态 `0x05E6`、电池 `0x05EB`、母线电压 `0x05EC`（0.1 V/字）、母线电流 `0x05F0`（0.1 A/字，含符号）、整车使能 `0x0608`（文本，使能显示 `R`）、采集确认 `0x0618`、存图状态 `0x0619`
模式键值	手动 `0x0010` / 遥控 `0x0008` / 示教 `0x0004` / 跟随 `0x0002` / 自动 `0x0001` / 采集 `0x0020`
档位键值	P `0x0008` / R `0x0004` / N `0x0002` / D `0x0001`
速度档键值	极低 `0x0008` = 35% / 低 `0x0004` = 60% / 中 `0x0002` = 80% / 高 `0x0001` = 100%

> ⚠ **屏能设档位，但只在 U7 没接的时候。** 一旦从 U7 解出过一次有效档位，实体开关就说了算， 屏不再能设档 —— 一个实体排挡杆压过触摸屏，方向是安全的。详见 §4.2。
> ⚠ 这一路**没有校验和**，也没有超时重发。屏没接上、波特率不对、线接反，现象都是「屏上不动」。

整车 CAN（CN34 / CN32）与两个 USB 串口

口	收发什么	详见
CN34 / CN32	经典 CAN，**1 Mbps**，标准帧，DLC = 8。本板发 `0x210`（20 ms），收 `0x211` / `0x212`（拓扑 B 另加 `0x213` / `0x214`）。终端 120 Ω 由 SW3 投切	**§5，逐字节给全**
USB1	只有 ASCII 文本：一行一条命令，回车换行结束；回复也是文本行	§2 – §3
USB2	只有二进制：22 字节定长帧，小端，`SOF = AA 55`，CRC16-MODBUS 覆盖第 2 – 19 字节	§6

1.3.3 出厂固件不使用的接口

这些口硬件是通的，出厂固件不碰，二次开发可以直接用（规格书 §17.3）。 列出来是为了**不让人对着一个接了线却没有反应的口去找原因。**

连接器	MCU 资源	状态
CN36	USART3（PC10 / PC11）	引脚已配成串口复用，但固件不初始化这一路，也不收发
CN33	I2C2（PB10 / PB11）	不初始化
H3	SPI2（PB12 – PB15）	不初始化
CN20 / CN19	PA2 / PA3，TIM2_CH3 / CH4	⚠ **轮速脉冲：固件在数，但计数没有出口** —— 不进 `tlm`、不进参数表、不上 CAN。从外面看等于没接。车速取自驱动板的 `0x212`
CN3 / CN4	TIM1_CH1 / CH2（PA8 / PE11）	PWM 已启动，但**占空比恒为 0** —— 行走电机由驱动板经 CAN 执行，本板不再输出 PWM
CN7 / CN10	TIM1_CH4 / CH3（PE14 / PE13）	未配成 PWM 通道
CN13 / CN14	TIM2_CH1 / CH2（PA0 / PA1）	捕获通道未启用

连接器	MCU 资源	状态
CN24 / CN25 / CN26 / CN29 / CN31	ADC1 另外五路	不采样。踏板固定用 CN30，换一路要重新编译
CN8 / CN9 / CN11 / CN12 / CN17 / CN18 / CN21	PE4 / PE7 / PE8 / PE9 / PE10 / PE12 / PE15	数字输出，**上电置低之后不再变化**
CN15 / CN16 / CN23 / CN27 / CN28	PD13 / PD12 / PD11 / PD9 / PD8	数字输入，固件不读
H2 · RC5 / RC6	PD14 / PD15，TIM4_CH3 / CH4	捕获通道未启用

1.3.4 两条容易踩的

一、CN5 / CN6 在规格书里叫「数字输出」，出厂固件把它们当输入用。 规格书 §8.1 把 PE2 / PE3 列进九路数字输出 —— 那是硬件的说法，没有错：这两个脚串了 100 Ω 就引到插座上，做输出做输入都通。出厂固件在初始化的最后一步把它们重配成**内部上拉输入**， 接刹车开关，踩下接 GND。

> ⇒ 照规格书把这两路当输出接了负载的人，得到的是一个悬空的上拉输入，负载不会动； 而刹车开关没有地方接。**接线以本节为准**，要拿它们做输出得改固件。

二、CN35 是 9600，不是 115200。 触摸屏那一路的波特率和两个 USB 口不同。拿串口工具去量这一路时按 9600 打开， 否则看到的是一串乱码，而现象和「屏没接上」一模一样。

---

2. 接上控制台

物理口	**USB1**（USART1）
串口参数	**115200 8N1，CR/LF**
内容	只有 ASCII 文本：命令进，回复出

2.1 ⚠ 打开串口之前必须知道的一件事

本板的 USB 转串口有一个交叉耦合的复位电路。**危险的不是任何一个静态电平，是一个组合：**

做了什么	结果（台架实测）
`{DTR=1, RTS=0}` 打开串口	应用照常运行
`{DTR=0, RTS=0}` 打开	也不复位
`{DTR=1, RTS=1}` 打开	也不复位
RTS 为低时做 DTR 跳变	也不复位
⚠ **RTS 拉高时做 DTR 跳变**	**应用下线，板子进入 ROM BootLoader**

推荐的打开方式是 `{DTR=1, RTS=0}`。 常见串口工具的默认行为都不会踢掉这块板， 但把 RTS 拉高之后再动 DTR 会。

> ⚠ **对一台装在车上的控制板，这件事的后果是它停发行驶请求。** 驱动器 300 ms 后超时停力矩， 而现场的人只看到「车不走了」，看不出板子进了 BootLoader。

怎么救回来（顺序错了会什么都不发生）：

```
{DTR=1, RTS=1}   保持 50 ms
{DTR=0, RTS=1}   保持 150 ms     ← 按住复位，RTS 必须还是高的
{DTR=1, RTS=0}   保持 300 ms     ← 松开复位，此时 BOOT0 已经是低
```

⚠ **不要先把电平恢复成安全值再去复位** —— RTS 为低时 DTR 跳变根本不产生复位， 于是「一次复位都没发生」，现象是什么都没变，而人会以为板子坏了。

2.2 上电先问的四条

```
ver          固件版本、构建标识、镜像校验、MCU 唯一 ID
tlm          一次遥测快照（六行，机器可读）
WHY?         现在为什么不允许驱动
ST?          驱动器链路状态、回环三态、各类计数器、复位原因
```

`ver` 的回答（0.1.1 起为 Release 构建、镜像已盖章）：

```
=tlm 1 build debug=0 stamped=1 crc=c77a92cb major=0 minor=1 patch=1 board=dlaboard_b scm=45568e0 cmdhash=...
DlaBoard_B 0.1.1 45568e0 built ...
firmware : 0.1.1 45568e0, crc ok
uid      : <24 位十六进制>
```

`id` 用一行说清这块板是什么，给上位机自动识别用。

2.3 遥测快照 `tlm`

七行 + `=tlm end`，全部是 `键=值`，单位写在名字里：

行	内容
`build`	版本、构建、盖章、命令表校验
`sys`	运行毫秒、复位原因、看门狗、HardFault 痕迹
`veh`	模式、档位、整车使能、刹车意图、踏板‰、速度档%、阻止位图
`spd`	车速 0.1 km/h、车速是否算数、行驶方向、母线电压 mV、母线电流 cA
`can`	链路三态（alive / frozen / believes）、健康字节、收发计数、发送失败、回显不符、短帧、FIFO 溢出、bus-off、刹车驱动位、拓扑
`steer`	转向执行器那一侧（0.1.4 起）： `0x231` 收帧数、链路活没活、帧龄、回显序列号、当前位置、状态位、故障码
`rc`	遥控链路三态、串口丢字节计数、转向拦截计数。⚠ `rc3=` 这一栏默认报的是 **iBUS** 链路

> **一趟主循环发一行。** 阻塞式地一次发完会让 `0x210` 停发，驱动器 300 ms 后超时。 上位机读取整张表请把超时放到 3 秒以上。

---

3. 命令一览

板上没有 `help`：这块板的命令表**随发布包发布**（`COMMANDS.json`，43 条）， 上位机据此画界面。板子在 `=tlm build cmdhash=` 里报这张表的 CRC32，**相等才用。**

组	命令
认板 / 遥测	`ver`（`VER?`）、`id`、`tlm`（`TLM?`）、`IMU?`、`JET?`
诊断	`WHY?`、`ST?`、`FOC ST?`、`GEAR?`、`RC?`、`BRK?`、`app`、`fault`、`fault codes`
参数	`param list`（`PARAM?`）、`param get <名>`、`param set <名> <值>`、`param save`、`param load`
标定	`calib list`、`calib status`、`calib start pedal`、`calib confirm`、`calib abort`
自检	`selftest run`、`selftest run all`
安全	`estop`、`clear`
台架驱动	`FOC D <%> [转向%]`、`FOC R <%> [转向%]`、`FOC RAW ...`、`FOC STOP`、`FOC OFF`、`BRAKE 1`、`BRAKE 0`、`BRAKE OFF`
台架注入	`FOC RX211 <8 个十六进制字节>`

> ⚠ `FOC D` / `FOC R` / `FOC RAW` **会让车走**。它们让控制台接管 `0x210`，并且在 `WHY?` 里置位 `cli_test`。台架命令 300 ms 不刷新就自动过期。

3.1 `app`：这块板上留给你改的那一层（0.1.9 起）

这块板的出厂固件从 0.1.9 起分成两层，而分层的目的只有一个：**让你能改整车怎么开，又不用碰安全那部分。**

层	里面是什么	你能不能改
核心层	输入采集、安全判断（什么时候允许驱动、什么时候强制刹车）、CAN 链路、看门狗、参数存储	❌
应用层 `App/apps/<名字>/`	各路输入怎么变成油门、转角、速度档；模式怎么切；多个节点怎么编排	✅ **就是给你改的**

源码在发布包的 `source.zip` 里（`App/apps/a1_diff2/` 是出厂那个应用，带 README）。改完**不用接车就能验**：

```
sh tests/sim/run.sh          喂一串输入，断言目标随时间怎么走
sh tests/run.sh              单元测试
```

板子上问它此刻算出来什么：

```
> app
APP name=a1_diff2 api=1 brake=0 act=1 band=60 out=60 clamp=0 L=350/0* R=350/0* steer=0/0*
```

字段	意思
`name` / `api`	现在编进去的是哪个应用、对着哪一版接口写的
`brake` / `act`	应用此刻的刹车意图 / 它认为该不该动。⚠ 这是**意见**，能不能动由核心层定
`band`	应用主张的速度档；`0` = 它不表态，用核心层算的那个
`out`	⭐ **真正发到 CAN 上的那个速度档**，见下
`clamp`	应用给过几次越界的数。一直不为 0 说明它的算术和量程对不上
`L=` / `R=` / `steer=`	各节点的 `牵引‰/转角‰`，`*` 表示这一拍应用没主张

> ⚠⚠ **今天只有速度档一个字段真的在线上**（0.1.10 起）。`out=` 就是这一帧 `0x210` 里发出去的那个字节；油门、挡位、转向仍由核心层那条老路算，`app` 里那几个数只是应用的意见。**改应用能改掉速度档，改不掉油门。** 这是有意为之：一次接一个字段，每接一个都要能证明"接之前和接之后线上一样"。
> ⚠ 控制台接管期间（`FOC RAW` / `FOC D` / `FOC R`）**不走应用层** —— 那几条命令的用途就是绕开整车仲裁发一帧指定内容。

规矩是：应用给 0，就用核心层算的那个。 所以出厂固件的行为和 0.1.8 完全一样（出厂应用把核心层给它的档原样传回去），而你把它改成别的数，线上就跟着变。

3.2 拿这块板做你自己的项目（0.1.11 起）

§3.1 说的是「**这台车**怎么开，那一层给你改」。这一节说的是另一件事：**你可以完全不要这台车。**

这块板是一块通用的整车控制器 —— 一颗 STM32F407、一路 CAN、三路串口、一组数字量输入和继电器输出、板载 EEPROM 和一颗六轴。出厂固件是我们拿它做的**一个最典型的示例**，不是你必须继承的地基。你要做一台完全不同的机器，从下面这个模板起步就行。

3.2.1 模板里有什么、没有什么

发布包的 `source.zip` 是**同一棵源码树**，里面有两个取舍：

取舍	编出来是什么
出厂固件（默认）	本书从 §4 起描述的那一整套：整车仲裁、档位、遥控、触摸屏、驱动器和转向那几帧
开发模板	**一块扶起来的空板子**：时钟外设、串口控制台、认板、参数存储、独立看门狗、镜像自述头。**主循环交给你，整车逻辑一行没有**

模板开箱就有的：

- 串口控制台，以及 `ver` / `id` 两条认板命令（上位机靠它认出你的板）
- 参数存储：板载 EEPROM，双区轮换 + CRC32 + 回读自验（§8）
- 独立看门狗（2 秒）、以及「上一次为什么复位」那份记录
- 镜像自述头（上位机按它认镜像）
- `param list` / `get` / `set` / `save` / `load` 五条命令，**回话和出厂固件逐字一样**
- CAN 外设已经起好，收发由你写

模板**没有**的：整车仲裁、档位解码、遥控、触摸屏、六轴、轮速，以及驱动器和 转向执行器的那几帧。要哪一样，去出厂固件那一侧把对应的文件拿过来。

3.2.2 怎么编、怎么烧

```
cmake --preset Release-minimal
cmake --build --preset Release-minimal
```

要能直接烧的 `.bin` / `.hex`（盖过章的）：

```
DLA_PROFILE=minimal DLA_CONFIG=Release sh scripts/build.sh
```

产物落在 `Release-minimal/`。

> ⚠ **产物和出厂固件分开放，别放同一个目录。** 两份不同的镜像同名摆在一起，烧错的那一次不会有任何东西报错 —— 板子只是从此变成了另一台机器。

你的代码写在 `App/core/app_minimal.c` 的两个函数里：`DlaCart_AppInit()`（开机一次）和 `DlaCart_AppLoopIteration()`（主循环一刻不停地调）。

> ⚠ **别在主循环里阻塞。** 看门狗是 2 秒，而喂狗那一句要等你的函数返回才会再执行 一次。要按固定周期做事，自己记时刻，别用延时函数。

这块板上有哪些引脚、哪些出厂固件没在用，见随包的 `pins.yaml`（⚠ 里面的「没在用」是**出厂固件没在用**，不是「随便用」—— 调试口和晶振那几脚 改掉，你的板子就烧不进去了）。

3.2.3 三样别删

模板对上位机的全部承诺就这三条。删掉任何一条，你的板子会在上位机里消失， 而**不会有任何一处告诉你是因为这里**：

别删	删了会怎样
看门狗	一个跑飞的处理器会留着最后那一刻的输出继续跑。这是这块板上最坏的一种失效，而它不报错 —— 该报错的那个东西自己已经跑飞了
`ver` / `id` 两条命令	上位机认板认的就是那几行话的措辞。改一个字，它会把你的板当成一块**被擦空的板**：面板上没有板名、型号选了也存不下、烧录和遥测跟着一起没
镜像自述头	上位机按它认镜像。差一个字节就完全认不出来，**而且是静默的** —— 它会退回去做整文件校验，看起来像「通过了」，于是一份坏镜像也能烧进去

3.2.4 哪些文件动之前要想一想

随包的 `template.yaml` 列着，分两组，**两组的分量不一样**：

组	说的是什么	举例
板级安全	动了**你自己的板子**会出事，或者出了事没人知道	看门狗；处理器故障之后、复位之前那个挂点；参数存储的读写和记录布局
工具链契约	动了**你的板子在我们这套工具里消失**	`ver` / `id` 那两条；build 记录那一行；镜像自述头；参数表的格式；板号；编译预设的名字

⚠ 这份名单很短，**这是有意的。**这个产品的意义就是让人改固件学硬件，所以默认可改， 名单上列的是例外。判据不是「重不重要」，是「**改错了会不会有人发现**」—— 会当场报错的东西不需要上名单。

> **一个挂点的例子。** 处理器故障（跑飞、访问了不该访问的地址）之后、板子复位之前， 固件会给你**最后一次**机会把会动的东西停下来。模板的默认实现是**空的**， 这是如实的：一块没接执行器的板子没有什么要停。 ⚠⚠ 你一旦开始驱动电机、继电器、气阀或者另一块板，**就该把它接上** —— 在你自己的任意一个源文件里写一个同名函数就行。不接也不会有任何东西报错： 板子照样复位，只是从故障发生到复位这一小段里，输出一直保持着故障前的样子。 具体怎么写，见 `App/Inc/board_isr_hooks.h` 文件头。

---

4. 整车控制逻辑

4.1 输入

输入	连接器	说明
踏板	**CN30**（ADC1_IN8）	端点由 `calib start pedal` 标定（§9），输出 0..1000‰
档位	**U7** 四位置选择开关	**脚 2=P、脚 3=R、脚 4=N、脚 5=D**（脚 1=GND）
整车使能	**CN22**（低有效）	未合 = 不允许驱动
刹车开关	**CN5 / CN6**（低有效，两路）	任一路踩下 = 刹车意图

输入	连接器	说明
遥控	**H2**（默认只用 RC3 走 iBUS）	iBUS：ch1 转向、ch2 油门、ch5/ch6 两个限速旋钮、ch7 急停。PWM 模式：RC1/RC2/RC4
速度档位	触摸屏（**CN35**）	0..100%，随 `0x210` 下发
车载计算机	**USB2** 二进制帧	FOLLOW / AUTO 模式

> 每一路收的是什么信号、什么格式、断了会怎样，见 §1.3.2； 固件不使用的那些口见 §1.3.3。

4.2 档位是电平解码，不是边沿闩锁

四路输入每一拍都解码成一个档位；**解不出（全高、或多路同时低）就落 P**， 并把两种情况分别计数（`gear_none_cnt` / `gear_ambig_cnt`）。

> 这里原来是「边沿闩锁」，代价是**线掉了就没有边沿，档位停在原地** —— 一台挂着 D 档、没有档位输入、而且没有任何东西发现的车。

`GEAR?` 把四路**原始电平**和解出的档位分开报，接线自检用。

4.3 刹车意图是「或」，不是优先级

物理刹车开关、遥控急停通道、档位在 P、整车使能断开 —— **任一路要求刹车即刹车。**

> ⚠ 这一条对台架有影响：只要本板在总线上断言刹车，驱动器那侧的控制台就清不掉它 （驱动器的合并里刹车也是「或」）。**验刹车极性要先把本板从 CAN 上摘掉。**

4.4 不活的源什么都不主张

「从来没接过」和「接过然后断了」必须是两个不同的答案：前者不是一个源， 后者是一个**失效的**源，要主张它的安全值。

- 遥控从来没接过 ⇒ 不主张任何东西（否则没插遥控的车永远刹着不动）；
- 遥控曾经活过、现在断了 ⇒ 摇杆交中位，REMOTE / COLLECT 下强制刹车。

4.5 指令源丢失：档位回 P，两块板一起撤销使能（0.1.4 起）

一个**曾经活过**的指令源断掉之后，本板不只是把它的主张清零，还会**闩住**：

```
档位 → P（空挡）        0x210 的 allow_drive → 0        0x230 的使能位 → 0
```

⇒ **链路回来，车不会自己接着走。** 要司机重新拨一次档才解除。 `tlm veh` 的阻止位图里会有一位说明现在是这个闩锁在挡着。

> ⚠ 这是一句对司机的承诺，不是一个可调参数。它存在的理由是另一种做法的样子： 一台车在链路断掉的几秒里保持着上一次的档位和油门，然后链路回来它接着跑 —— 而这几秒里司机以为它已经停了。

哪些算"断"：遥控 / 采集，接收机曾有信号后 200 ms 无帧；自动模式，上位机指令帧超 300 ms； 跟随模式，跟随帧超 200 ms。手动 / 训练没有远端源，不算断。遥控急停通道另算。

4.6 司机清故障的三个动作（0.1.7 起）

驱动板或转向执行器锁存了故障之后，司机不用连串口 —— 三个动作里任意一个都算一次清故障：

动作	判据
连拨两次 P	1.5 秒之内回到 P 两次

动作	判据
长按 N	在 N 挡上按住 2 秒（一次按住只算一次）
屏幕按钮	屏幕往地址 `0x0620` 写 1

一个动作清整车：本板自己的急停闩锁 + 转向执行器（`0x230` 的清故障位举一帧）+ 驱动板（`0x210` 挡位字节 bit7 的上升沿）。`tlm veh` 末尾的 `clrs=` 是累计次数。

> ⚠ 屏幕那个按钮的键在屏幕 UI 工程里还没有画，固件这一侧是齐的。

---

5. CAN 总线接口（整车侧）

本章是自足的：按本章接线、按本章解析，就能和这块板通信，不需要读别的章节。

> **字节布局的权威副本是随源码发布的 `App/Inc/foc_can.h`（转向那两帧是 `App/Inc/steer_can.h`）。** 本章与它冲突时以它为准，那说明本章是需要修的。⚠ 0.1.4 之前这两个文件在 `Core/Inc/` 下，0.1.4 把手写代码整体搬进了 `App/`。

5.1 总线层

项目	规格
协议	**经典 CAN（不是 CAN FD）**
波特率	**1 Mbps**
帧格式	**标准帧（11 位 ID）**，全部 **DLC = 8**
字节序	**小端**（多字节字段低字节在前）
物理层	ISO 11898-2，3.3 V 收发器
终端电阻	板载 120 Ω 经拨动开关投切。**总线只在两端各投一个**，中间节点不投
接收滤波	本板 **mask = 0，全收**，再按 ID 分发
DLC 检查	本板消费的每一帧都要求 DLC = 8，**短帧计数并丢弃**（`ST?` 的 `shortrx`）

5.2 ID 分配

ID	方向	周期	内容	本章
`0x210`	控制板 → **广播**	**20 ms**	行驶请求。所有驱动板都收	§5.4
`0x211`	驱动板 → 控制板	20 ms	状态（双电机板；或拓扑 B 的左轮板）	§5.5
`0x212`	驱动板 → 控制板	100 ms	母线遥测 + 车速	§5.6
`0x213`	右轮板 → 控制板	20 ms	状态（**仅拓扑 B**）	§5.7
`0x214`	右轮板 → 控制板	100 ms	遥测（**仅拓扑 B**）	§5.7
`0x230`	控制板 → 转向执行器	**20 ms**	转向指令（0.1.4 起）	§5.10
`0x231`	转向执行器 → 控制板	20 ms	转向状态（0.1.4 起）	§5.10

> ⚠ **给总线上新增节点时，不要占用 `0x213` / `0x214`。** 它们是拓扑 B 的右轮板，现在没接不等于以后不接；占了它，那一天的现象是两个节点发同一个 ID、内容不同。

两种驱动拓扑，帧格式完全一致：

拓扑	组成	谁发什么
A	一块双电机驱动板	`0x211`（同时填左右两个轮速槽）+ `0x212`
B	两块单电机驱动板	左板 `0x211`（只填左轮槽）+ `0x212`；右板 `0x213` + `0x214`

本板把拓扑 B 两块板的上行**合并成与双电机板等价的视图**（轮速分槽、健康位各管半边、 母线电流求和）。**拓扑靠「帧从哪来」判定，不靠「某个值恰好是不是 0」** —— 见 §5.7 的注意事项。

5.3 一眼速查

```
0x210  控制板发  20 ms
  [0:1] u16 油门 ‰ 0..1000        [2:3] i16 转向 ‰ -1000..1000（本板恒发 0）
  [4]   标志：b0 允许驱动 b1 刹车 b2-7 速度档（2% 一档，0=不声明）
  [5]   挡位 0=N 1=前进 2=后退    [6] 序列号（每帧自增）
  [7]   超时 x10 ms（本板发 30 = 300 ms）

0x211  驱动板发  20 ms
  [0]   健康位（全部 1 = 好）      [1] 驱动板序列号
  [2:3] i16 左轮车架转速 rpm       [4:5] i16 右轮车架转速 rpm
  [6]   回显它解出的整个标志字节   [7] b7 命令有效 / b6 本地锁 / b0-3 采信的挡位

0x212  驱动板发  100 ms
  [0:1] u16 母线电压 mV            [2:3] i16 母线电流 10 mA/LSB
  [4]   序列号                     [5:6] u16 车速 0.1 km/h（绝对值）
  [7]   保留 0

0x230  控制板发  20 ms      转向指令（0.1.4 起）
  [0:1] i16 目标：‰ -1000..1000，或 mm x10（当 [2] b1 置位）
  [2]   标志：b0 使能 b1 单位=mm b2 回中 b3 清故障（**上升沿**触发）
  [3]   速度限制 0..255 = 对端配置上限的 0..100%（0 = 用对端默认）
  [4]   序列号（每帧自增，状态帧里回显）
  [5]   指令超时 x10 ms（0 = 用对端默认 300 ms）   [6][7] 保留 0

0x231  转向执行器发  20 ms
  [0:1] i16 当前位置 ‰             [2:3] i16 当前位置 mm x10
  [4]   状态位 b0 已标定 b1 已使能 b2 到位 b3 在负限位 b4 在正限位
        b5 转向故障（码在 [5]） b6 助力 b7 **驱动侧**故障锁存（桥关了）
  [5]   转向故障码                 [6] i8 电机电流 Iq A x10
  [7]   序列号回显
```

5.4 `0x210` 行驶请求（控制板 → 驱动板）

周期 20 ms，永不停发。 帧是周期帧，**安全状态写在帧的内容里，不靠停发表达。**

字节	类型	字段	取值
`[0:1]`	u16 LE	油门	0..1000（‰）
`[2:3]`	i16 LE	转向	−1000..+1000（‰）。⚠ **本板恒发 0**，见 §5.10
`[4]`	位域	标志	见下表
`[5]`	u8	挡位	0 = N / 1 = 前进 / 2 = 后退（>2 一律当 0）

字节	类型	字段	取值
`[6]`	u8	序列号	每帧自增，用来证明发送方还活着
`[7]`	u8	超时	×10 ms。0 = 用驱动器默认值；本板发 **30**（300 ms）

`[4]` 标志字节：

位	含义
bit0	`allow_drive` —— 允许驱动
bit1	`brake` —— 请求刹车
bit2 – bit7	**速度档**，u6，**2% 一档**，0..50 → 0..100%

速度档的三条规矩（最容易搞错的地方）：

① `0` 是「不声明」，不是「档位为零」。 驱动器收到 0 时不主张这个字段， 沿用它上一次收到的档位。**发送方也不要把 0 当有效值发出去。**

计算方式：`档位步数 = ceil(百分比 / 2)`，写进 `[4] >> 2`。反向解码：`百分比 = 步数 × 2`。

② 发的是百分比，不是档位序号，而且下游不许再乘一次。 速度档说的是**能跑多快**，不是**能出多大力**。它到了驱动器那边只作用于**转速上限**， 不缩放扭矩 —— 最低档和最高档一样有劲，只是顶速低。 如果消费端又把它乘进油门，就乘了两次，表现为低速档带不动人。

③ 2% 的量化误差是写下来的，不是抹掉的。 四个常用档 35 / 60 / 80 / 100 里只有 35% 落不准， 到驱动器是 36%（250 rpm 上限上的 2.5 rpm，约 0.12 km/h）。**取整方向是向上** —— 宁可比屏上写的高一个最小步长，也不要让车比司机以为的还慢。

一帧完整的例子（油门 350‰、不转向、允许驱动、不刹车、速度档 60%、前进、序列号 0x2A、超时 300 ms）：

```
ID 0x210  DLC 8
5E 01 00 00 79 01 2A 1E
|__|__|  |__|__|  |  |  |
 350‰    转向 0    |  |  +-- 30 x 10 ms = 300 ms
                  |  |  +----- 序列号 0x2A
                  |  +-------- 挡位 1 = 前进
                  +----------- 标志 0x79 = 0b01111001
                               b0=1 允许驱动, b1=0 不刹车,
                               b2-7 = 0b011110 = 30 步 x 2% = 60%
```

5.5 `0x211` 驱动板状态（驱动板 → 控制板）

字节	内容
`[0]`	健康位，见下表
`[1]`	**驱动板自己的序列号** —— 判它活没活的依据
`[2:3]`	i16 LE 左轮**车架**机械转速 rpm（饱和处理）
`[4:5]`	i16 LE 右轮同上
`[6]`	**回显它最后一次解出的整个标志字节**（含速度档六位）
`[7]`	b7 = 它认为我们这一路还活着；b6 = 本地锁；b0 – b3 = 它最后采信的挡位

`[0]` 健康字节 —— 这个字节各位一律「1 = 好」：

位	含义
bit0	电机 1 运行中
bit1	电机 2 运行中
bit2	电机 1 霍尔新鲜
bit3	电机 2 霍尔新鲜
bit4	电机 1 栅驱无故障
bit5	电机 2 栅驱无故障
bit6	最近收到过命令
bit7	**刹车驱动已使能**

> ⚠ **「1 = 好」是一条归一约定，不是照抄引脚电平。** 驱动器上低有效的故障信号在发送前 已经归一成 `_ok`。解析这个字节时不要再按引脚极性反一次。

`[7]` 字节 —— ⚠ 这个字节不适用「1 = 好」的约定：

位	含义
bit7	命令有效：驱动器认为控制板这一路还活着
bit6	**本地锁：1 = 驱动器处于本地模式**，它照收照回，但不按总线动作
bit3 – bit0	它最后采信的挡位

`[6]` / `[7]` 是回环，不是装饰。 它们让控制板能分开三件本来长得一样的事：

现象	含义	去查哪根线
帧不来了	**对方根本没听见我**	线、接头、对端有没有电
帧在来，`[1]` 序列号不动	**对方卡住了**（应用挂死，邮箱还在重传）	对端固件
帧在来，`[7]` bit7 = 0	**听见了，但已经不信我了**	我们的发送侧、序列号有没有在动

本板三条都判（`ST?` 的 `alive` / `frozen` / `believes`）。

> ⚠ **回显 `[6]` 只比对 `allow_drive` 和 `brake` 两位。** 速度档六位在改档那一帧和它回显之间 必然差一拍，比了会一直报错；而这两位是安全语义。

5.6 `0x212` 母线遥测（驱动板 → 控制板）

字节	内容
`[0:1]`	u16 LE 母线电压，**mV**
`[2:3]`	i16 LE 母线电流，**10 mA/LSB**（拓扑 B 下两块板求和）
`[4]`	序列号
`[5:6]`	u16 LE **车速，0.1 km/h**，绝对值，夹到 9999
`[7]`	保留，0

车速为什么由驱动器换算好了再发，而不是消费端自己拿转速乘轮周长：

转速属于电机，和轮胎无关，所以 `0x211` 发转速是对的。但每一个想要车速的消费者都得乘一个 轮周长，于是每一个都自己长了一份。2026-08-25 首次真总线联调时，同一个物理量在这台车上 有**三个家**：驱动器 flash 里 798 mm

（滚一圈量地面的），控制板两处各一个 816 mm （都是从标称直径算的）。后果不只是显示偏高 2.26% —— 自动驾驶的速度闭环用的是同一个偏高的 反馈量，于是它会稳定在比目标低 2.26% 的速度上。

⇒ **换算只有一个家：那块被告知过轮周长、并把它存在自己 flash 里的驱动板。** 换个轮子改一条命令，不用重编译，也不用通知任何消费者。

> ⚠ `[5:6]` **读到 0 有两个意思**：车停住了，和线上根本没有车速（旧固件不填这个字段）。 本板用三条证据区分：遥测帧新不新鲜；车速报 0 而轮子在转；以及量级 —— 超过 60 km/h 的 读数不是车速，是坏帧。三条任一不过，整条车速反馈判为 **不算数**，自动驾驶不会使能。
> 这一条要紧，是因为自动驾驶是速度闭环：把「没数据」当成「0 km/h」，误差每 20 ms 都等于 整个目标值，积分器会绕到上限，油门钉死在开环加速上。

方向不在这一帧里。 `0x212` 的车速是绝对值，行驶方向要看 `0x211` 里带符号的轮速。

5.6b 上行两帧的解码示例

同一时刻收到的两帧，逐字节读一遍：

```
ID 0x211  DLC 8
70 3C 78 00 88 FF 79 81

[0] = 0x70 = 0b01110000
        bit4 电机1 栅驱正常 = 1     bit5 电机2 栅驱正常 = 1
        bit6 最近收到过命令 = 1     bit7 刹车驱动已使能 = 0  <- 见 §5.9 ①
        bit0/1 两路都没在跑，bit2/3 霍尔不新鲜（车停着，正常）
[1] = 0x3C  驱动板序列号 60。与上一帧比对：变了 = 它没卡住
[2:3] = 78 00 (小端) -> 0x0078 = +120 rpm   左轮（车架坐标系，正 = 前进）
[4:5] = 88 FF (小端) -> 0xFF88 = -120 rpm   右轮
        ⚠ 一正一负是正常的：双轮车必有一侧反装。两侧同号才要查
[6] = 0x79  它回显我们上一帧的标志字节：与我们发的 0x79 一致
        ⚠ 只比 bit0/bit1；速度档六位差一拍是正常的
[7] = 0x81 = 0b10000001
        bit7 命令有效 = 1  它认我们这一路
        bit6 本地锁  = 0  ⚠ 0 不等于「没锁」，见 §5.9 ②
        bit0-3 采信的挡位 = 1（前进）

ID 0x212  DLC 8
7A 94 FA 00 11 2D 00 00

[0:1] = 7A 94 (小端) -> 0x947A = 38010 mV = 38.0 V   母线电压
[2:3] = FA 00 (小端) -> 0x00FA = 250 x 10 mA = 2.50 A 母线电流
[4]   = 0x11  遥测序列号
[5:6] = 2D 00 (小端) -> 0x002D = 45 x 0.1 = 4.5 km/h  车速（绝对值）
[7]   = 0x00  保留
```

同一帧数据能得出的结论： 链路健康（序列号在动、它认我们）、车在以 4.5 km/h 前进、 母线 38.0 V / 2.5 A、**刹车驱动没有使能**（`0x211[0]` bit7 = 0 —— 但这一位的 0 是有歧义的， 不要直接报「刹车故障」）。

> ⚠⚠ **两个协议里的挡位编码不一样，不要互相套用：**

在哪	编码
CAN `0x210[5]` 与 `0x211[7]` 低四位	**0 = N，1 = 前进，2 = 后退**
车载计算机串口帧（§6）	**0 = P，1 = R，2 = N，3 = D**

两者各自与对面约定，谁也不能改。做网关的人请显式转换，不要直接透传这个字节。

5.7 拓扑 B：`0x213` / `0x214`

格式与 `0x211` / `0x212` **完全一致**，只是发送方是右轮板，且只填自己那半边。

本板的合并规则：

量	合并方式
轮速	各填各的槽
健康位 bit0 – bit5	每侧管自己那三位
健康位 bit6 / bit7	**两块板都会写**（它们是关于整车的判断），两帧的原始字节各留一份，按访问器的规则合并；刹车驱动位取**或**
母线电流	求和

⚠ **拓扑判定要攒够证据。** 本板连收 3 帧 `0x213` 才闩定为拓扑 B（闩定后只有断电才复位）。 代价是真接两块单板时，**头 60 ms 仍按拓扑 A 解释** `0x211`，而拓扑 B 的左板不填右轮槽。 第一次搭拓扑 B 时，请先看这 60 ms 有没有产生假读数。

5.8 时序、超时与失效行为

链路	判据	超时	到期后
`0x210` 发送	—	20 ms 周期	**永不停发**；安全状态写在内容里
`0x211` 到达	最后一帧的时刻	**500 ms**	判为不新鲜 ⇒ 阻止驱动
`0x211` 序列号	`[1]` 动没动	**300 ms**（= 15 帧）	判为对端卡住 ⇒ 阻止驱动
`0x212` 到达	最后一帧的时刻	**500 ms**	车速反馈不算数 ⇒ 自动驾驶撤防
车速合理性	`[5:6]` ≤ 600（60 km/h）	每帧	超出即判坏帧
车载计算机 AUTO `0x11`	帧新鲜度	**300 ms**	刹车 + 速度 0 + 转向回中
车载计算机 FOLLOW `0x10`	帧新鲜度	**200 ms**	刹车 + 油门 0
遥控	帧新鲜度	**200 ms**	摇杆交中位；REMOTE / COLLECT 下强制刹车
遥控急停通道	帧新鲜度	200 ms	**曾活过而断了 ⇒ 刹车**；从来没接过 ⇒ 不主张
触摸屏	心跳	8 s	仅影响指示灯，不影响控制

本板挂掉时会怎样： `0x210` 停发，驱动器 300 ms 后超时停力矩 —— 但**它无从知道该刹车**，因为刹车意图写在那一帧的一个位里。本板有 2 秒的独立看门狗， 挂死会自行复位；HardFault 会立即闩死急停并复位（不再是停在原地保持最后状态）。

5.9 三个「这一位说的比它看起来要小」

① `0x211[0]` bit7 = 刹车驱动已使能。

它精确说的是驱动器里**一个被存下来的决定**，**不是一次测量**。没有任何地方验证过刹车执行器 真的接着。

措辞	能推出	验证过吗
刹车驱动使能开着	我发的刹车位会走到 H 桥	✅ 这就是这个标志本身
BRK 口上驱动着一个执行器	**车真的会停**	❌ 没有

执行器掉了、线断了、卡死了，这一位照样是 1。⇒ **它不构成「本车可以停下」的证据。**

反过来，**0 有两个意思**（这台车没装刹车执行器 / 装了但没使能），总线上分不开。 **不要把 0 直接显示成「刹车故障」。** 早期驱动器固件该位恒为 0，与「未使能」同形。

② `0x211[7]` bit6 = 本地锁，0 是有歧义的。

本地锁在驱动器上从 0.4.8 就有，但 **0.4.8 / 0.4.9 / 0.4.10 无论锁没锁都发 0**， 从 0.4.11 起才如实上报。而这条总线上没有任何字段携带对端固件版本。

⇒ **0 的意思是「没锁，或者对端固件比 0.4.11 老」。** 在成员不全是 0.4.11+ 的总线上，不许把 0 读成「没锁」。 本板因此**只在读到 1 的时候动作**：1 永远不是谎话（不实现它的固件发 0），0 什么都不主张。

③ 为什么本地锁必须单独一位、不能从别处推。

被锁的驱动板和完全服从的驱动板，在这一位出现之前**是同一帧**：线掉了的板会停发 `0x211`， 看得见；而被锁的板照发、照报命令有效、照回显标志、照置「最近收到过命令」—— 因为那四样回答的都是「帧在不在、新不新鲜」，而那**仍然是真的。** 锁住发生在它更上面的一层。⇒ 没有这一位，控制板会从三个诚实地回答着错误问题的位， 使能一台不会听话的机器。

5.10 转向：`0x230` / `0x231`（0.1.4 起接通）

> ⚠ 本节 Rev 1.1 及更早写的是「转向：现在是断的」。**那句话从固件 0.1.4 起不再成立** —— 2026-09-09 在真车上实测走通：司机打方向 → 控制板发 `0x230` → 齿条跟着走。

`0x210[2:3]` **转向字段仍然恒发 0，而且以后也是 0。** 转向不走行驶请求帧，走自己的 `0x230`：一条独立的指令帧，一条独立的状态帧。这样转向执行器不必收行驶请求， 也就不会被行驶请求里的任何一位意外武装。

本板发 `0x230` 的规矩：

- **无条件每 20 ms 发一帧**，链路活不活由对端按帧到不到判断，不由本板决定发不发。
- **使能位不是调用方给的**，它由发送函数自己推导：整车使能（PD10）为真、且已经有人 设过目标、且这一帧不在清故障 —— 三者与起来才置位。⇒ 调用方**没有办法**把这一位传错。
- **还没有人设过目标之前，使能位恒 0**：帧照发（告诉对端"我还在，别动"），轴不上电。 ⇒ 上层实现了一半也是安全的。
- 手动 / 训练两个模式会**显式放开**转向（`SteerCan_Release`），因为那时人在手动转。 它不会自己超时落回 —— 一个"一段时间没人设目标就断电"的机制，在一次主循环卡顿里 就是方向机掉电。
- 五个用到转向的模式统一用 **‰ 目标**（±1000 映射到对端的软限位），不发 mm。
- `[3]` 速度限制、`[5]` 超时的覆盖值**只作用于本帧**，不写进对端的持久化配置。

⚠⚠ `[2]` **b3 清故障是按这一位自己的上升沿触发的，不是电平。** `[4]` 序列号每帧自增，所以一个"盯着序列号变化"的边沿检测器等于电平检测器 —— 一个一直把 b3 举着的控制器会每 20 ms 清一次故障，而越程故障在齿条静止之后不会重新锁存， **翻圈保护就真的没有了，而且没有任何地方会说出来。**同一帧里同时带「使能 + 清故障」 也不会上电：要先放掉 b3，再发使能，两帧 —— 这本来就是"一次新的使能"的意思。

收 `0x231` 之后本板做的事： 位置进 `tlm` 的 `steer` 行；`[4]` b7（驱动侧故障锁存） 和 b5（转向故障）是两件事，前者是"桥关了、齿条根本不会动"，后者是"轴不相信自己的位置"。

> ⚠⚠ **不要把牵引驱动板的固件烧进转向执行器**（反过来也一样）。两者从同一套代码分出来， 烧错了它会正常启动、正常在串口上说话、通过每一道校验 —— 而齿条没有任何限位， 并且会无条件发 `0x211` / `0x212`，和真的驱动板撞 ID。 认固件只认发布包 `MANIFEST.txt` 里的 `name:`，不要按目录名或"哪一版最新"挑。

装反了怎么办（0.1.8 起）： 参数 `steer_invert`，默认 0。这套转向系统的 安装方式决定了绝大多数情况下方向是固定的（**执行器正方向 = 车辆右转**），装车流程里 没有"设左右"这一步；只有极少数改变用途、执行器装反了的车才把它置 1 —— 置 1 之后 `0x230` 的目标在线上取反、`0x231` 回来的位置和左右限位镜像，本板内部一律 仍按"正 = 右转"。执行器那一侧什么都不改。

5.11 第三方接入清单

要消费这块板的行驶请求（做一个驱动器）：

1. 收 `0x210`，DLC = 8，按 §5.4 解析；
2. 用 `[6]` 序列号判发送方活没活，**不要只判「帧到没到」**；
3. 用 `[7]` 的超时值做自己的看门狗（本板发 300 ms）；
4. `[4]` bit2-7 为 0 时**沿用上一次的速度档**，不要当成 0%；
5. 速度档只作用于转速上限，**不要乘进扭矩**；
6. 周期上报 `0x211`（20 ms）与 `0x212`（100 ms），**并且如实填 `[6]` / `[7]` 回环字段** —— 那是控制板区分三种失效的唯一依据；
7. 健康字节按「1 = 好」归一，没装的东西如实报 0。

要旁听这条总线（做一个仪表、记录仪或诊断工具）：

1. 只收不发，不要投终端电阻（除非你在总线物理末端）；
2. 车速取 `0x212[5:6]`，**不要自己拿 `0x211` 的转速乘轮周长**；
3. 方向取 `0x211` 轮速的符号；
4. 判断链路健康用序列号，不用帧计数；
5. `0x211[0]` bit7 与 `0x211[7]` bit6 的 0 都不要当成结论（§5.9）。

不要做的：

- 不要占用 `0x213` / `0x214`；
- 不要在 `0x210` 里加位、改位 —— CAN 的字节位置是两块板共有的，**判据不是「加还是改」，而是两侧的定义有没有一起改**；
- 不要在总线上放第二个 `0x210` 的发送方。

5.12 没有整车也能测

台架上没有驱动器时，用这三条把 CAN 这一侧走通：

```
FOC RAW <油门‰> <转向> <标志> <挡位> [超时x10ms]      让控制台接管 0x210 发一帧指定内容
FOC RX211 <b0> <b1> ... <b7>                          往本板注入一帧 0x211（十六进制）
ST?                                                    看解析结果、回环三态与全部计数器
```

`FOC RX211` 是验证解析侧的正规手段：本地锁位就是这么验的（灌 `0 → 1 → 0` 三个方向）。 ⚠ 但**它只验解析这一半** —— 「驱动器真的会发这一位」是另一件事，要两块板挂在同一条总线上才算数。

`ST?` 里的 `tx210` 一行打印**本板最后一帧实际发出的内容**。加这一行是因为 「急停会覆盖发出去的那一帧」这条规则，在总线上没有驱动器的时候无法被观测 —— 而一条观测不到的安全规则，从外面看和没有那条规则一样。

---

6. 与车载计算机的串口协议

物理口	**USB2**（UART5），115200 8N1
帧	**22 字节定长**，小端
帧头	`AA 55`，版本 `01`

校验	**CRC16-MODBUS**，覆盖偏移 2 – 19，放在末尾

类型	方向	用途
`0x10`	计算机 → 板	跟随模式指令
`0x11`	计算机 → 板	**自动驾驶指令**
`0x02`	板 → 计算机	跟随就绪 / 状态
`0x03`	板 → 计算机	**自动驾驶就绪 / 状态**，约 10 Hz
`0x01` / `0x04` / `0x05` / `0x06` / `0x07`	板 → 计算机	控制、心跳、手动、遥控、采集上报

`0x11` 自动驾驶指令的载荷：

字段	类型	说明
目标转角	i16，0.1°	左负右正
目标车速	i16，mm/s	+ 前进 / − 后退 / 0 停
刹车	u8	1 = 主动刹车，优先于目标车速
挡位	u8	0=P 1=R 2=N 3=D，回显 / 校验用
标志	u8	bit0 = 请求使能

> 即使还没有使能，计算机也应当**持续发 `0x11` 心跳**（速度 0、刹车 1 等安全值 + 置请求使能位），板子据此加上自身的安全条件来决定使不使能。**300 ms 收不到有效帧就刹车 + 速度归零 + 清积分。**

`0x03` 状态帧的载荷：

字段	类型	说明
实测车速	i16，mm/s	+ 前进 / − 后退
实测转向	i16，0.1°	
状态位	u8	bit0 已使能 / bit1 驱动器状态新鲜 / bit2 整车使能 / bit3 急停或接管 / bit4 处于自动模式
挡位	u8	
阻止位图	u16	§7 的 `drive_block` 位图，0 = 没有任何一条在挡着
固件版本	u8 + u8	高 4 位主版本 / 低 4 位次版本；再一字节修订号

> 阻止位图与版本占用的是原先写死为 0 的四个保留字节。**这是加字段不是改含义** —— 老的消费端把它们当保留、读到 0，现在读到非 0 也只会继续忽略。
>
> 加它的理由：自动驾驶期间 `0x03` 是唯一的上行通道，而「为什么不允许驱动」恰恰是那时最需要 问出来的东西。⚠ **这四个字节的内容尚未在真车上核对过**（见 §13），接入时请先核一帧实帧。

使能条件（全部满足才使能）： 屏上选中自动模式 + 驱动器状态新鲜 + 整车使能合 + 档位 D + `0x11` 心跳在 300 ms 内 + 计算机请求使能 + 无急停 / 无接管。任一不满足即不使能，并强制刹车。

---

7. 为什么不允许驱动

这块板刻意提供的能力：整车不走的原因可能在使能开关、档位、刹车、遥控、CAN 对端等任意一处，**从外面看现象完全一样。** `WHY?` 把它们分开报。

```
WHY?          位图 + 短标识符，例如：block=0x0003 ven,gear
fault         当前生效的条目，带一句「这是什么」和一句「下一步查哪儿」
fault codes   完整的十四条表
```

位	标识符	含义	性质
bit0	`ven`	整车使能开关未合	活值
bit1	`gear`	档位不在 D / R（自动模式下要求 D）	活值
bit2	`brake_sw`	刹车踏板开关按下	活值
bit3	`rc_brake`	遥控急停通道在刹车位	活值
bit4	`rc_lost`	遥控链路曾活过、现已断（REMOTE / COLLECT 下）	活值
bit5	`foc_stale`	驱动器状态帧不新鲜（> 500 ms）	活值
bit6	`speed_invalid`	车速反馈不算数（没数据 / 字段空 / 量级不合理）	活值
bit7	`host_stale`	车载计算机指令帧不新鲜	活值
bit8	`host_disable`	车载计算机没有请求使能	活值
bit9	`cli_test`	台架命令正在接管行驶请求	活值
bit10	`foc_frozen`	帧在来但序列号不动 —— 驱动器应用卡住了	活值
bit11	`foc_disbelief`	驱动器回说它已不认我们这一路	活值
bit12	`foc_locked`	驱动器报告它在本地模式（只对 1 反应，见 §5.9）	活值
bit13	`estop`	**急停闩锁**	**锁存**，要一条 `clear`

> **位只增不重用。** 上位机把人话、图、警告挂在位号上，重编号会静默地换掉一整屏解释。
> ⚠ bit12（驱动器本地锁）在**任何模式**下都成立 —— 被锁的驱动器不会因为司机在踩踏板就听话。

---

8. 参数

```
param list            整张表：名字、单位、范围、默认值、当前值、可读可写、要不要存
param get <名>
param set <名> <值>
param save            写进板载 EEPROM，并立刻按下次开机的方式回读自验
param load            读回
```

表分两类：

类	举例	可写
设置（id 1 起）	自动驾驶限速与增益、各类超时、踏板端点与死区、遥控阈值、CAN 周期与新鲜度门限、档位去抖	✅ 可写、可存
活值（id 60 起）	模式、档位、整车使能、刹车意图、踏板‰、速度档、阻止位图、车速、母线电压电流、链路三态、各类计数器、复位原因	只读

三条使用规矩：

一、行数以帧里的 `count=` 为准，不要写死在你的客户端里 —— 加一行是常事。

二、`param set` 回显成功不等于命令生效。 每次改完都读回来。本板遵守一条硬规矩： **一个参数报「可写」，当且仅当真正执行它的那段代码读的是运行时值。** 宁可少几个可写参数，也不要一个说了不算的「可写」。

三、参数存在板外 EEPROM，重新烧录固件不会擦掉它。 存储格式为双区轮换 + CRC32：两个区交替写，两区都读、校验通过且序号大的赢 —— 任何时刻至少有一个完整的旧记录还在，掉电写坏也回得来。 只允许追加字段，旧记录按自己的长度校验通过，缺的取默认值。

四、每条记录写明自己属于哪一张参数表（0.1.11 起）。 上面第三条有个直接后果，值得单独说：**一条记录会活到下一份烧进去的固件里去， 哪怕那份固件的参数表完全不一样。** 而参数编号是各表自己编的 —— 出厂固件的 1 号是自动模式限速（0 – 1500 mm/s），开发模板（§3.2）的 1 号是那个示例参数（−1000 – 1000）。 不加区分的话，在模板上存一个 42、再烧回出厂固件，那个 42 会**如实地**变成 42 mm/s 的自动限速：编号在、可写、数值也在范围内，三道校验全过， 而没有任何一处会说一句话。

⇒ 现在记录头里带着「这是哪张表写的」，号对不上就**整条不用**，参数全部保持默认， 并且 `param load` 会把这件事说出口。

> ⚠⚠ **它挡住误读，挡不住覆盖。** 两个备份区是轮换写的，所以在另一份固件上 **存两次**，就把两个区都写成它的了 —— 先存的那一份从此没了。 ⇒ **在两份固件之间来回烧之前，先 `param list` 把要紧的值抄一份出来**（尤其是标定过的踏板端点）。

8.1 读不到参数时，板子会说清楚是哪一种（0.1.12 起）

`param load` 报「装载了 0 条」有四个完全不同的原因，而它们要查的东西不一样。 从 0.1.12 起每一种各有一句话：

板子说的	意思	该做什么
`nothing has ever been saved on this board`	从来没存过	出厂状态，**不是故障**
`a record is there but does not check out`	有记录但校验不过	重新 `param set` + `param save`
`the stored record belongs to a DIFFERENT parameter table`	这块板上烧过另一份固件，它存的参数不属于当前这张表	见上面第四条
`no storage chip answered on I2C1`	存储芯片没应答	**先拔一次电再上电**，见下

> ⭐ **「刚才还好好的，现在说没有存储芯片」——拔一次电。** 处理器可以在任何一刻被复位（烧录、看门狗、按复位键），包括正从存储芯片读一个字节 读到一半的时候。那一刻芯片正把数据线拉着，要等下一个时钟才放手；复位之后 处理器忘了这件事，而那条线一直被拽着低 —— 总线上从此谁都问不出话来， **从外面看和「没焊存储芯片」一模一样。** 0.1.12 起固件开机会先把总线放开，所以这种情况基本不会再出现； 万一遇到，断电重上电一定能解。

---

9. 踏板标定

```
calib list          这块板有哪些标定项
calib start pedal   开始踏板端点标定，板子一步步给提示
calib confirm       确认当前这一步（松开 / 踩到底）
calib status        现在走到第几步
calib abort         放弃
```

标定的是踏板的两个端点（松开位与踩到底位），结果落进参数表并 `param save` 保存。 **上位机照着 `calib status` 一步步带用户走**，每一步都有机器可读的回执。

---

10. 自检

```
selftest run          八项，逐条给结论
```

输出是机器可读的：`=st 1 result item=<项> verdict=<pass|fail|info|skip> ...`，最后一行 `=st 1 summary total= pass= fail=`。

项	判什么	说明
`image`	镜像自述头与校验	
`eeprom_present`	I²C 上有没有 EEPROM 应答	器件在不在
`eeprom_rw`	对**隔离地址**的读写	不碰参数存储的两个区
`param_record`	存储里有没有一条有效记录	`info` —— 出厂的板子本来就是空的，不是故障
`imu`	惯性传感器在不在	
`pedal_adc`	踏板 ADC 当前读数	`info` —— 这里没有「应该是多少」
`watchdog_bit`	看门狗咬过没有	`info` —— 它说的是过去发生过，而板子此刻是好的
`can_loopback`	—	**`skip`**，并说明理由：环回会把这块板从总线上摘掉、停发行驶请求

> 自检只做「读已有状态」和「对隔离资源的读写」，**不重配任何正在服役的外设** —— 一块正在发行驶请求的板子，自检不该把它自己的通信摘掉。

---

11. 急停与安全申报

11.1 急停

```
estop          立即闩死：行驶请求被压成「停 + 刹车」。任何状态下都可调
clear          显式解除（边沿触发，一条命令 = 一次事件）
```

这条规则写在**行驶请求发送函数的内部**，不是写在它的调用点上 —— 本板的行驶请求今天有两个 出口（台架命令与正常仲裁），明天可能有第三个；放进被调用的一侧，新出口天生就受它管。

11.2 ⚠ 安全申报（必读）

问题	本板的答案
有没有**不经过 CPU** 的关断通路?	❌ **没有。** 本板没有功率级、没有栅极驱动，它的急停是**软件的**：闩死一个标志，让每一帧行驶请求变成「停 + 刹车」。
有没有温度保护?	❌ **没有。** 本板不测温度。母线电压电流来自驱动器的遥测帧，是**转述**，不是本板测的。
载人 / 载重场景是否需要外部物理断路器?	⚠⚠ **需要，而且是强制的。** 本板的急停依赖 MCU 在跑、CAN 在通、驱动器在响应。任何失控会造成伤害的场合，必须另配独立的物理断路器，**直接切断驱动器的动力回路，不经过本板。**

另外两条与安全相关的实现：

- **HardFault**：不再停在原地保持最后状态，而是闩死急停 → 在不清零的 RAM 里留下痕迹 → 立即复位。痕迹在 `tlm` 与 `fault` 里报得出来（⚠ 它活过复位，但**活不过断电**）。
- **独立看门狗 2 秒**：挂死会自行复位。

---

12. 升级与发布包

引导方式	**本板没有 bootloader，也没有 A/B 槽** —— 只有一个镜像，从 `0x08000000` 开始
升级通道	MCU 的 ROM BootLoader（ISP），经 USB1
进入方式	§2.1 的 RTS / DTR 时序
镜像自述头	位于镜像偏移 `0x200`：长度、CRC32、版本、构建标识、是否 Debug 构建

发布包目录内含：`MANIFEST.txt`、`.bin`、`.hex`、`SHA256SUMS`、`COMMANDS.json`。

发布包还带 `source.zip`：**按 git 跟踪文件打的整棵源码树**，出厂固件和开发模板 （§3.2）都从它编出来，主机测试和仿真也在里面。

0.1.13 的 `MANIFEST.txt` 摘要：

```
name: DlaBoard_B
version: 0.1.13
commit: 6d7c7d9
built: 2026-09-11T04:26:29Z
config: Release
application: DlaBoard_B.bin 62000 bytes, crc32 9fae03e9, load 0x08000000
source: source.zip
```

> ⚠ 角色词是 `application:` 而不是 `slot A:` —— **这块板没有槽。** 0.1.6 之前写的是 `slot A:`，那是迁就上位机当时的词表，而一块没有槽的板自称 有槽，任何将来对「槽」做推断的东西（A/B 升级、回滚）都会得出错误结论。
> ⚠ **升级期间这块板停发行驶请求。** 对一台装在车上的控制板，请确保车辆处于安全状态再升级。

升级之后先做三件事：

1. `ver` —— 确认版本号和 `crc ok`；
2. `param list` —— 确认参数还在（参数在板外 EEPROM 上，烧录不会动它）。 ⚠ 如果这块板上换过另一份固件，先读一遍 §8 第四条；
3. **把板子上接着的每样东西各问一句** —— 屏（`app` 的 `out=` 应该是屏上选的那个档）、 驱动器（`tlm can` 的 `alive=1`）、存储芯片（`selftest run`）。 ⚠ 判据不是「板子没死」，是**每一路都给出它该给的那个数。**烧录会复位这块板， 但不会复位接在它上面的东西，所以刚烧完那一刻最容易看出一路没接上。

---

13. 已验 / 未验（如实列出）

⚠ 这张表的每一行都是刻意分开的。合并任何两行，都会让某个没验过的东西看起来验过了。

已在板上验证：

项	说明
安全项、看门狗、构建标识	含反向验证：临时挂死后确实由看门狗复位
`WHY?` 阻止位图	台架上报出 `ven,gear`（使能断开 + 档位在 P）
CAN 回环三态（alive / frozen / believes）	
参数表读取、写入、存盘与回读自验	
`0x211[0]` bit7 刹车驱动使能	**两个方向都由接收端观测**：置位读到 1，关回读到 0
健康位合并（**拓扑 A**）	

项	说明
串口 DTR / RTS 的危险组合与救回序列	§2.1 全部为实测
发布包整链	探测 → 全片擦除 → 写入 → 回读校验 → 复位到应用 → `ver` 有回应
转向链路 `0x230` / `0x231`	2026-09-09 **真车实测走通**（固件 0.1.4 起），见 §5.10

尚未验证（请勿当作保证）：

项	为什么还没验
刹车驱动使能位的**冷启动**行为	已验的那次对侧走的是调试器复位，不是拔电
健康位合并（**拓扑 B**）	手上只有一块双电机驱动板，无板可验
`0x211[7]` bit6 本地锁的**线上**行为	**解码已验**（注入 `0→1→0` 三个方向都对）；「驱动器真的会发」没验
急停对已发出帧的**覆盖作用**	台架上使能开关是断开的，仲裁已经把同样的字段压成了安全值 ⇒ 覆盖在这里观测不到。要验必须合上使能开关（**车轮离地**）
`0x03` 状态帧的阻止位图与版本字节	自动模式只能从触摸屏进入，台架板没接屏
档位接线	**标准已定、空载已验**（不接线时落 P、计数器不涨）；**接上真开关拨一遍杆**没做
刹车执行器极性	若极性反了，「刹车」命令做的是松刹车。⚠ 验极性要**先把本板从 CAN 上摘掉**（见 §4.3）
轮速脉冲输入（CN20 / CN19）	固件在数脉冲，但计数**没有出口**，也没在车上标定过（哪一路对应左轮还没确定）。今天没有任何东西依赖它：车速走 `0x212`，见 §1.3.3
转向正负号参数 `steer_invert`	参数和主机测试都有（0.1.8），但**没有一台装反的车验过**。默认 0，不改行为，见 §5.10 末

14. 版本历史

14.1 固件

版本	日期	内容
0.1.13	2026-09-11	🔴 **修：接了触摸屏的车，屏一直连不上**（0.1.11 / 0.1.12 受影响，0.1.10 及更早没有）。现象是屏上的档位、速度、模式不再刷新，屏上选的速度档也不再发给驱动器（车还能开，只是速度档这一项不再由屏说了算）。原因：板子靠每 2 秒向屏问一句来确认屏还在，而那一句写在了「已经确认屏在」的分支里 —— 屏不会自己开口，它只回答问题，于是一旦板子认为屏不在，它就再也不问。**装了屏的车建议从 0.1.11 / 0.1.12 升上来；没装屏的车不受影响。**
0.1.12	2026-09-11	修：从服务器取下源码之后，跑一遍主机测试会红，而红的原因不是你的代码（一份"改之前想一想"的名单点了一个编译时才生成的文件，源码包里没有它）。⚠ 固件行为和 0.1.11 完全一样。另：`param load` 读不到参数时说清楚是哪一种（§8.1）；引脚核对那一行在中文 Windows 上不再是乱码
0.1.11	2026-09-11	⭐ **这块板可以拿来做你自己的项目了**（§3.2）：同一棵源码树多了一个「开发模板」取舍，编出来的固件把板子扶起来、主循环交给你，整车逻辑一行没有。随包多了一份 `template.yaml`（哪些文件动之前要想一想，分板级安全 / 工具链契约两组）。⚠ 出厂固件的行为一个字节没变：这一版对它做的是内部整理，控制台回的每一句话都逐条比对过。另有两处修复：**参数记录现在写明属于哪一张参数表**（§8 第四条 —— 在此之前，换固件之后一边存的值会被另一边当成另一个参数读进去，而三道校验全过、没有一处报错）；**连着烧几次固件之后存储芯片有时会「消失」**，开机现在会先把总线放开（§8.1）
0.1.10	2026-09-10	**应用层的第一个字段真的上线：速度档**（§3.1）。规矩是「应用给 0 就用核心层算的那个」，所以出厂固件的行为和 0.1.8 完全一样，而改了应用线上就跟着变。⚠ 控制台接管（`FOC RAW` / `FOC D` / `FOC R`）那一支不走应用层

版本	日期	内容
0.1.9	2026-09-10	**出厂固件分层**：核心层 / 应用层（`App/apps/`），新增控制台命令 `app`，主机仿真 `sh tests/sim/run.sh`。这一版应用层只算不发。另外定下多节点「逐轮指令」的 CAN 契约（`0x232+n` / `0x240+n`），给以后四轮各自驱动、各自转向用；这一版线上一帧都不发
0.1.8	2026-09-10	新增参数 `steer_invert`（0/1，默认 0）：执行器装反了的车在本板翻转向正负号，`0x230` 的目标线上取反、`0x231` 的位置和左右限位镜像，本板内部一律仍按「正 = 右转」（§5.10 末）
0.1.7	2026-09-10	**司机清故障的三个动作**（连拨两次 P / 长按 N 2 s / 屏幕按钮），一个动作清整车（本板急停闩锁 + 转向 + 驱动板）；`tlm veh` 末尾加 `clrs=`（§4.6）
0.1.6	2026-09-10	发布包带 `source.zip`（按 git 跟踪文件打，可复现）；清单头补 `commit:` / `built:`
0.1.5	2026-09-10	四个处理器故障向量（HardFault / MemManage / BusFault / UsageFault）都走同一条"闩死急停 → 留痕 → 复位"；`fault` 的故障码成帧输出；发布默认 Release 构建
0.1.4	2026-09-10	⭐ 这一版变化最大：**转向接上 `0x230` / `0x231`**（§5.10，09-09 真车实测走通）；**指令源丢失闩锁**（档位回 P、两块板一起撤销使能，§4.5）；**遥控从 PWM 改 iBUS**（10 通道，急停 SwA=ch7、限速 VrA=ch5），两个旋钮分开管油门上限和转向上限；删掉第三方伺服（`servo_can`）整个模块；`tlm` 加一组 `steer`；构建改 CMake + Ninja，手写代码搬进 `App/`（`Core/` 交还 CubeMX）
0.1.3	2026-09-07	**自复位挡杆**（`gear_sw_type`）—— 外接的挡杆大多是自复位的
0.1.2	2026-09-07	**挂 D / R 之后头两帧压零油门** —— 驱动板有一条"踩着油门不许挂 D"的互锁，而这块板把挡位字节当成油门方向位算，那台车因此永远进不了 D 挡
0.1.1	2026-09-01	`id` 命令；发布包 `MANIFEST` 的 CRC32 算法修正
0.1.0	2026-08-31	首个 **Release** 构建；镜像自述头与盖章；`SHA256SUMS`；命令表 id 唯一性守卫；随固件出厂的自检；踏板标定编排；`estop` / `clear` 与 HardFault 处理；参数持久化（板载 EEPROM，双区轮换 + CRC32 + 回读自验）；`fault` / `fault codes`；`param get` / `set` / `save`；档位电平解码与接线标准；驱动器本地锁解码；CLI 与车载计算机分线到两个 USB 口

14.2 本书

版本	日期	说明
Rev. 1.6	2026-09-11	跟上固件 0.1.11 – 0.1.13。新增 **§3.2「拿这块板做你自己的项目」** —— 开发模板怎么编怎么烧、三样别删、哪些文件动之前要想一想；§8 补第四条「每条记录写明属于哪一张参数表」与那条挡不住的**覆盖**（来回烧两份固件之前先把参数抄出来），新增 **§8.1「读不到参数时板子会说清楚是哪一种」**（含「刚才还好好的，现在说没有存储芯片 → 拔一次电」）；§12 的 `MANIFEST.txt` 示例从 0.1.1 换成现行版（角色词也从 `slot A:` 订正为 `application:`，那是 0.1.6 就改了的），升级后的自检从两件加到三件（**接着的每样东西各问一句**）。⚠ 另订正两处**书里自相矛盾**、都不是固件变了的地方：① **页脚**写着「固件 0.1.8 · 文档 Rev. 1.3」，而扉页写的是 0.1.10 / Rev. 1.5 —— 同一本书三个答案；② §15 已知限制还写着「转向未接通」，而 §5.10 和 §13 都写着 2026-09-09 真车走通。这两处和 Rev. 1.4 修的是同一类
Rev. 1.5	2026-09-10	跟上固件 0.1.9 / 0.1.10。新增 **§3.1 `app`：这块板上留给你改的那一层** —— 出厂固件分成核心层和应用层，应用层（`App/apps/`，源码在发布包的 `source.zip` 里）就是给你改的，改完不接车也能验。⚠ 那一节把「**今天只有速度档一个字段真的在线上**」写在最显眼的地方：改应用能改掉速度档、改不掉油门，而且控制台接管期间不走应用层。§3 命令表 42→43 条，诊断那一组加 `app`
Rev. 1.4	2026-09-10	订正两处**书里自相矛盾**的地方。两处都是 0.1.4 那一版改遥控和转向时留下的旧文字，**不是固件变了**：① §13「尚未验证」里那行「转向 | 整条链路未接通」，而它指向的 §5.10 写的是 2026-09-09 真车走通 ⇒ 转向链路进「已验证」表，未验的那条改成 `steer_invert` 参数（没有一台装反的车验过）。② §1.3.1 接口表里 **RC4 有两行、互相矛盾**，第二行「只采集、不参与刹车逻辑」是 0.1.4 以前的接法（那时急停在 RC3）⇒ 删掉；`App/Inc/rc_input.h` 里同源的旧注释一并订正，那份更要紧——照它去 RC3 上量急停，量到的是一串 iBUS 数据

版本	日期	说明
Rev. 1.3	2026-09-10	跟上 0.1.8：`steer_invert` 那段去掉「尚未发布」
Rev. 1.2	2026-09-10	跟上固件 0.1.2–0.1.7（书此前停在 0.1.1，落后六版）。§5.10 从「转向：现在是断的」改写成 `0x230` / `0x231` 的全文——那句话从 0.1.4 起就不成立了；§5.2 / §5.3 补上转向两帧；新增 §4.5 指令源丢失闩锁、§4.6 司机清故障的三个动作；§2.3 补 `tlm` 的 `steer` 行；契约头路径 `Core/Inc/` 订正为 `App/Inc/`
Rev. 1.1	2026-09-04	新增 §1.3 接口功能定义：逐个连接器写出出厂固件的用途、每一路收的是什么信号与什么格式（踏板 0 – 5 V 模拟量、干接点、舵机 PWM、触摸屏帧格式与全部地址键值），以及固件不使用的那些口；§4.1 输入表补上连接器
Rev. 1.0	2026-09-02	首版，对应固件 0.1.1。此前固件内容以《产品规格书》Rev. 1.1 §17 的预览形式发布，自本书起独立成册。CAN 协议自本版起在本书 §5 完整给出

> ⚠ 规格书 Rev. 1.1 §17.3 写着「参数只读、不可掉电保存」与「出厂为 Debug 构建」，**这两条自 0.1.0 起已不成立**（参数可写可存，出厂为 Release 构建并已盖章）。以本书为准。

15. 已知限制（固件侧）

#	项	说明
1	**急停是软件的**	本板无不经过 CPU 的关断通路。载人 / 载重场合必须另配物理断路器，见 §11.2
2	**转向正负号没有在装反的车上验过**	参数 `steer_invert` 和主机测试都有（0.1.8 起），但没有一台装反的车验过。默认 0，不改行为，见 §5.10 末。⚠ 本行 Rev. 1.6 订正：原来写的是「转向未接通、`0x210[2:3]` 恒发 0」，那是 0.1.4 之前的事实，而 §5.10 与 §13 都写着 2026-09-09 真车走通 —— 同一本书里的两个答案
3	**无温度保护**	本板不测温度；母线读数是驱动器的转述
4	**命令表的 `needs` 字段全填 `none`**	上位机的词表是给带功率级的板子设计的（桥 / 力矩 / 已标定），本板一条都不成立。本板真正的前提是「使能开关合 + 挂 D 档」，即 §7 的阻止位图。**这是已知的失真，不是漏填**
5	**台架命令会接管行驶请求**	`FOC D` / `FOC R` / `FOC RAW` 期间 `WHY?` 会显示 `cli_test`；命令 300 ms 不刷新即过期
6	**拓扑 B 切换有 60 ms 窗口**	连收 3 帧 `0x213` 才闩定，其间按拓扑 A 解释，见 §5.7
7	**在两份固件之间来回烧会丢参数**	记录带了表号之后不会再被**误读**，但备份区是轮换写的，挡不住**覆盖** —— 在另一份固件上存两次就把两个区都占了。换固件前先 `param list` 抄一份，见 §8 第四条
8	**开发模板没有在客户的机器上走过一遍**	§3.2 那条路在我们这边从公网下载的包上验过（起工程 → 编 → 主机测试 → 烧 → 认板 → 参数存取活过复位），但还没有客户从零走过。遇到对不上的地方请告诉我们