TC STM32F446 - Code

tags: `Traction Control` `RWD` `STM32F446` `CAN` `Steering Angle`

---

0. ???????

0.1 ??

- 車輛：後驅賽車，電動或引擎皆可（此處假設有「驅動扭矩命令」介面）。
- 晶片：STM32F446（180 MHz, FPU）。
- 功能：
    - 四輪輪速 + IMU + 方向盤轉角 + 馬達扭矩 → 即時計算後輪 slip。
    - 使用方向盤轉角為主的 Target Slip（避免 IMU 雜訊）。
    - PI 控制後輪扭矩（只限扭，不主動煞車）。
    - 三種模式：直線 / 繞 8 / 賽道；只允許在「靜止」時切換。
    - 感測器錯誤偵測 → TC 自動關閉（Fail-safe）。
    - 積分重置邏輯（Anti-windup）。
    - 簡易路面 μ 估測 → 自適應 Kp。

0.2 ??????????

- `Core/Src/main.c`：系統進入點、TIM/CAN 中斷。
- `Core/Inc/tc_types.h`：共用資料結構與 enum。
- `Core/Inc/tc_controller.h` / `Core/Src/tc_controller.c`：TC 核心演算法。
- `Core/Inc/tc_fault.h` / `Core/Src/tc_fault.c`：感測器錯誤偵測 & 硬體安全。
- `Core/Inc/tc_mode.h` / `Core/Src/tc_mode.c`：模式管理（直線/繞 8/賽道）。
- `Core/Inc/tc_io.h` / `Core/Src/tc_io.c`：CAN 解析 / 封包 + 按鈕處理。
- `Core/Inc/tc_params.h`：可調參數（Kp, Ki, slip 目標等）。

---

1. ??????????tc_types.h?

```
#ifndef TC_TYPES_H
#define TC_TYPES_H


#include "stdint.h"
```

```
#include "stdbool.h"

/* ---- 模式 ---- */
typedef enum {
    TC_MODE_STRAIGHT = 0,  // 直線加速
    TC_MODE_FIGURE8  = 1,  // 繞 8 / skidpad
    TC_MODE_TRACK    = 2   // 賽道
} TC_Mode_t;

/* ---- TC 狀態 ---- */
typedef enum {
    TC_STATUS_OFF      = 0,  // 關閉
    TC_STATUS_NORMAL   = 1,  // 正常控制
    TC_STATUS_SAFETY   = 2,  // 安全限扭
    TC_STATUS_FAULT    = 3   // 錯誤關閉
} TC_Status_t;

/* ---- 錯誤旗標 ---- */
typedef enum {
    FAULT_NONE         = 0,
    FAULT_WHEEL        = 1 << 0,
    FAULT_IMU          = 1 << 1,
    FAULT_STEER        = 1 << 2,
    FAULT_CAN_TIMEOUT  = 1 << 3,
    FAULT_TORQUE       = 1 << 4
} TC_FaultFlags_t;

/* ---- 單次迴圈輸入（10 ms） ---- */
typedef struct {
    /* 原始感測器 + 指令（從 CAN 或 ADC 來） */
    float omega_fl, omega_fr;     // 前輪輪速 rad/s
    float omega_rl, omega_rr;     // 後輪輪速 rad/s
    float ax_imu;                 // 縱向加速度 m/s2
    float ay_imu;                 // 橫向加速度（目前可不必用）
    float psi_dot;                // yaw rate rad/s
    float steer_deg_raw;          // 方向盤轉角 deg（未濾波）
    float T_driver;               // 駕駛要求扭矩 Nm

    /* 系統時間（ms） */
    uint32_t timestamp_ms;
```

```
} TC_Input_t;

/* ---- 單次迴圈輸出 ---- */
typedef struct {
    float v_mps;                  // 車速 m/s
    float slip_L, slip_R, slip_avg;
    float steer_deg_f;            // 濾波後方向盤 deg
    float lambda_target;          // 目標 slip
    float delta_T;                // PI 輸出比例（無單位）
    float T_cmd;                  // 實際下發扭矩 Nm
    TC_Status_t tc_status;
    uint32_t fault_flags;
} TC_Output_t;

#endif
```

2. ??????????tc_params.h?

```
#ifndef TC_PARAMS_H
#define TC_PARAMS_H

/* 輪胎半徑 */
#define R_FRONT        0.32f
#define R_REAR         0.32f

/* TC 主迴圈週期 */
#define TC_DT_SEC      0.01f  // 10 ms, 100 Hz

/* 方向盤濾波係數（低通） */
#define STEER_ALPHA    0.92f  // 越接近 1 越慢，約 3 Hz

/* Slip 安全閾值 */
#define SLIP_SINGLE_MAX 0.40f   // 單輪 > 0.4 視為嚴重打滑

/* 低速關閉 TC 門檻 */
#define V_TC_OFF_KMH   3.0f

/* 積分上限（Anti-windup） */
```

```
#define INTEGRAL_MAX   1.0f

/* ---- 三種模式基線參數 ---- */

/* 直線模式 */
#define KP_STRAIGHT    4.2f
#define KI_STRAIGHT    0.35f
#define LAMBDA_BASE_STRAIGHT  0.20f
#define STEER_SENS_STRAIGHT   0.4f   // 方向盤敏感度較低
#define MIN_TQ_RATIO_STRAIGHT 0.55f  // 最小 55% 扭矩

/* 繞 8 模式 */
#define KP_FIGURE8     3.1f
#define KI_FIGURE8     0.42f
#define LAMBDA_BASE_F8 0.16f
#define STEER_SENS_F8  1.0f
#define MIN_TQ_RATIO_F8 0.40f

/* 賽道模式 */
#define KP_TRACK       2.8f
#define KI_TRACK       0.50f
#define LAMBDA_BASE_TRACK 0.14f
#define STEER_SENS_TRACK  1.3f
#define MIN_TQ_RATIO_TRACK 0.30f

#endif
```

---

3. ?????tc_mode.h / tc_mode.c?

3.1 ????tc_mode.h?

```
#ifndef TC_MODE_H
#define TC_MODE_H

#include "tc_types.h"
#include "tc_params.h"
```

```
typedef struct {
    TC_Mode_t current_mode;   // 目前實際模式
    TC_Mode_t target_mode;    // 車手選擇的模式
    bool      change_pending; // 是否等待靜止後切換

    float Kp, Ki;
    float lambda_base;
    float steer_sens;
    float min_tq_ratio;       // 最低扭矩比例
} TC_ModeState_t;

void TC_Mode_Init(TC_ModeState_t *m);
void TC_Mode_RequestChange(TC_ModeState_t *m, TC_Mode_t new_mode, float v_kmh);
void TC_Mode_ApplyIfStopped(TC_ModeState_t *m, float v_kmh);

#endif
```

3.2 ???tc_mode.c?

```
#include "tc_mode.h"

static void TC_Mode_ApplyParams(TC_ModeState_t *m, TC_Mode_t mode) {
    m->current_mode = mode;

    switch (mode) {
    case TC_MODE_STRAIGHT:
        m->Kp          = KP_STRAIGHT;
        m->Ki          = KI_STRAIGHT;
        m->lambda_base = LAMBDA_BASE_STRAIGHT;
        m->steer_sens  = STEER_SENS_STRAIGHT;
        m->min_tq_ratio= MIN_TQ_RATIO_STRAIGHT;
        break;

    case TC_MODE_FIGURE8:
        m->Kp          = KP_FIGURE8;
        m->Ki          = KI_FIGURE8;
        m->lambda_base = LAMBDA_BASE_F8;
        m->steer_sens  = STEER_SENS_F8;
        m->min_tq_ratio= MIN_TQ_RATIO_F8;
```

```
        break;

    case TC_MODE_TRACK:
    default:
        m->Kp          = KP_TRACK;
        m->Ki          = KI_TRACK;
        m->lambda_base = LAMBDA_BASE_TRACK;
        m->steer_sens  = STEER_SENS_TRACK;
        m->min_tq_ratio= MIN_TQ_RATIO_TRACK;
        break;
    }
}

void TC_Mode_Init(TC_ModeState_t *m) {
    m->current_mode   = TC_MODE_FIGURE8;   // 建議預設繞 8 模式
    m->target_mode    = m->current_mode;
    m->change_pending = false;
    TC_Mode_ApplyParams(m, m->current_mode);
}

/* 車手按按鈕後呼叫: 若靜止就立即切, 若在跑就設成 pending */
void TC_Mode_RequestChange(TC_ModeState_t *m, TC_Mode_t new_mode, float v_kmh) {
    m->target_mode = new_mode;

    if (v_kmh < 1.0f) { // < 1 km/h 視為靜止, 直接切
        TC_Mode_ApplyParams(m, new_mode);
        m->change_pending = false;
    } else {
        m->change_pending = true;
    }
}

/* 每次迴圈呼叫: 若 pending 且已靜止 → 套用新模式 */
void TC_Mode_ApplyIfStopped(TC_ModeState_t *m, float v_kmh) {
    if (m->change_pending && v_kmh < 1.0f) {
        TC_Mode_ApplyParams(m, m->target_mode);
        m->change_pending = false;
    }
}
```

4. ??????????tc_fault.h / tc_fault.c?

4.1 ????tc_fault.h?

```
#ifndef TC_FAULT_H
#define TC_FAULT_H

#include "tc_types.h"

typedef struct {
    uint32_t flags;
    uint32_t wheel_err_count;
    uint32_t imu_err_count;
    uint32_t steer_err_count;
    uint32_t torque_err_count;
    uint32_t can_timeout_count;

    float    omega_prev[4];
    float    steer_prev;
} TC_FaultState_t;

void TC_Fault_Init(TC_FaultState_t *f);
void TC_Fault_Update(const TC_Input_t *in, TC_FaultState_t *f);
bool TC_Fault_IsSafeToRun(const TC_FaultState_t *f);

#endif
```

4.2 ???tc_fault.c?

```
#include "tc_fault.h"
#include "math.h"

#define OMEGA_MIN      -1.0f
#define OMEGA_MAX      200.0f
#define OMEGA_JUMP_MAX  50.0f
```

```
#define AX_MAX         20.0f
#define STEER_JUMP_MAX 15.0f   // deg/10ms

#define FAULT_COUNT_MAX 3

void TC_Fault_Init(TC_FaultState_t *f) {
    f->flags = FAULT_NONE;
    f->wheel_err_count = 0;
    f->imu_err_count   = 0;
    f->steer_err_count = 0;
    f->torque_err_count= 0;
    f->can_timeout_count = 0;
    for (int i = 0; i < 4; i++) f->omega_prev[i] = 0.0f;
    f->steer_prev = 0.0f;
}

void TC_Fault_Update(const TC_Input_t *in, TC_FaultState_t *f) {
    /* 1. □□□□ + □□ */
    float omegas[4] = { in->omega_fl, in->omega_fr, in->omega_rl, in->omega_rr };
    bool wheel_fault = false;

    for (int i = 0; i < 4; i++) {
        if (omegas[i] < OMEGA_MIN || omegas[i] > OMEGA_MAX) {
            wheel_fault = true;
        }
        float diff = fabsf(omegas[i] - f->omega_prev[i]);
        if (diff > OMEGA_JUMP_MAX) {
            wheel_fault = true;
        }
        f->omega_prev[i] = omegas[i];
    }

    if (wheel_fault) {
        if (++f->wheel_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_WHEEL;
        }
    } else {
        f->wheel_err_count = 0;
    }
```

```
    /* 2. IMU 範圍 */
    if (fabsf(in->ax_imu) > AX_MAX) {
        if (++f->imu_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_IMU;
        }
    } else {
        f->imu_err_count = 0;
    }

    /* 3. 方向盤跳動 */
    float steer_diff = fabsf(in->steer_deg_raw - f->steer_prev);
    if (steer_diff > STEER_JUMP_MAX) {
        if (++f->steer_err_count >= FAULT_COUNT_MAX) {
            f->flags |= FAULT_STEER;
        }
    } else {
        f->steer_err_count = 0;
    }
    f->steer_prev = in->steer_deg_raw;

    /* 4. 扭矩範圍（依車輛實際調整上限） */
    if (in->T_driver < 0.0f || in->T_driver > 1000.0f) {
        f->flags |= FAULT_TORQUE;
    }
}

bool TC_Fault_IsSafeToRun(const TC_FaultState_t *f) {
    /* 有關鍵錯誤就不允許跑 TC */
    if (f->flags & (FAULT_WHEEL | FAULT_TORQUE | FAULT_CAN_TIMEOUT)) {
        return false;
    }
    return true;
}
```

5. TC ?????tc_controller.h / tc_controller.c?

5.1 ????tc_controller.h?

```
#ifndef TC_CONTROLLER_H
#define TC_CONTROLLER_H

#include "tc_types.h"
#include "tc_mode.h"

typedef struct {
    /* 內部狀態 */
    float integral;          // PI 積分
    float v_imu_est;         // 從 IMU 積分的速度估計
    float steer_deg_f;       // 濾波後方向盤 deg
    float ax_peak;           // 用於 μ 估測的加速度峰值

    uint32_t integral_reset_count;
} TC_Internal_t;

void TC_Controller_Init(TC_Internal_t *s);
void TC_Controller_Update(
    const TC_Input_t  *in,
    const TC_ModeState_t *mode,
    TC_Internal_t     *state,
    TC_Output_t       *out
);

#endif
```

5.2 ??????

1. 由前輪輪速估算車速 (v)。
2. 後輪 slip ratio: (\lambda_L = (\omega_{RL} R_{rear} - v) / v)，(\lambda_R) 類似。
3. 方向盤轉角低通濾波。
4. 根據模式參數 + 濾波後轉角 + 車速，算出 Target Slip (\lambda^*)。
5. 計算誤差 (e = \lambda^* - \bar{\lambda})。
6. PI 控制（含積分 Anti-windup + 重置邏輯）。
7. 安全邏輯（低速、嚴重 slip、錯誤）覆蓋一切。
8. 最後輸出 T_cmd。

5.3 ???tc_controller.c?

```
#include "tc_controller.h"
#include "tc_params.h"
#include "math.h"

void TC_Controller_Init(TC_Internal_t *s) {
    s->integral = 0.0f;
    s->v_imu_est = 0.0f;
    s->steer_deg_f = 0.0f;
    s->ax_peak = 0.0f;
    s->integral_reset_count = 0;
}

/* 車速估計（簡化版: 先只用前輪） */
static float TC_EstimateSpeed(const TC_Input_t *in) {
    float v = (in->omega_fl + in->omega_fr) * 0.5f * R_FRONT;
    if (v < 0.1f) v = 0.1f;
    return v;
}

/* Slip Ratio */
static void TC_ComputeSlip(const TC_Input_t *in, float v,
                           float *slip_L, float *slip_R, float *slip_avg) {
    float v_safe = (v > 0.1f) ? v : 0.1f;
    *slip_L = (in->omega_rl * R_REAR - v_safe) / v_safe;
    *slip_R = (in->omega_rr * R_REAR - v_safe) / v_safe;
    *slip_avg = (*slip_L + *slip_R) * 0.5f;
}

/* 方向盤低通濾波 */
static float TC_FilterSteer(float raw_deg, TC_Internal_t *s) {
    s->steer_deg_f = STEER_ALPHA * s->steer_deg_f +
                     (1.0f - STEER_ALPHA) * raw_deg;
    return s->steer_deg_f;
}

/* 簡易 μ 估測，用 ax 峰值分級 → 回傳倍數 0.5~1.0 */
static float TC_EstimateMu(TC_Internal_t *s, const TC_Input_t *in) {
    float ax_abs = fabsf(in->ax_imu);
    /* 指數平滑追蹤峰值 */
    s->ax_peak = fmaxf(s->ax_peak * 0.95f + ax_abs * 0.05f, ax_abs);
```

```
    if (s->ax_peak > 8.5f)   return 1.0f;  // 高 μ
    if (s->ax_peak > 5.5f)   return 0.8f;  // 中 μ
    if (s->ax_peak > 3.5f)   return 0.65f; // 低 μ
    return 0.5f;
}

/* 方向盤為主的 Target Slip（模式 + steer_sens） */
static float TC_ComputeTargetSlip(
    float steer_deg_f,
    float v,
    const TC_ModeState_t *mode)
{
    float d = fabsf(steer_deg_f);
    float k_steer;

    /* 分段線性: 可把斜率乘上 mode->steer_sens 調整靈敏度 */
    if (d < 4.0f) {
        k_steer = 1.0f;
    } else if (d < 20.0f) {
        float slope = 0.018f * mode->steer_sens;
        k_steer = 1.0f - slope * (d - 4.0f);    // 直到中彎
    } else {
        float slope2 = 0.01f * mode->steer_sens;
        k_steer = 0.65f - slope2 * (d - 20.0f);
    }

    if (k_steer < 0.4f) k_steer = 0.4f;

    /* 速度補償（高速略放寬） */
    float speed_factor = fminf(v * 3.6f / 40.0f, 1.0f) * 0.03f + 0.97f;

    float lambda = mode->lambda_base * k_steer * speed_factor;
    if (lambda < 0.08f) lambda = 0.08f;
    return lambda;
}

/* 積分重置邏輯（你之前討論的「防飽和」） */
static void TC_IntegralResetLogic(const TC_Input_t *in,
                                  const TC_ModeState_t *mode,
```

```
                                    TC_Internal_t *s,
                                    float v_mps)
{
    float v_kmh = v_mps * 3.6f;

    /* 1) 低速: 直接清零, 起步最安全 */
    if (v_kmh < 2.0f) {
        if (s->integral != 0.0f) {
            s->integral = 0.0f;
            s->integral_reset_count++;
        }
        return;
    }

    /* 2) 駕駛幾乎沒踩油門: 快速衰減 */
    if (fabsf(in->T_driver) < 2.0f) {
        s->integral *= 0.1f;
        return;
    }

    /* 3) 模式剛剛切換（因為你只在靜止時切換, 這裡可省略或保留） */
    (void)mode; // 目前不需要特別判斷, 保留介面
}

/* 主控制: 不含 Safety（Safety 放在 main.c） */
void TC_Controller_Update(
    const TC_Input_t  *in,
    const TC_ModeState_t *mode,
    TC_Internal_t     *state,
    TC_Output_t       *out)
{
    /* 1. 車速 & slip */
    out->v_mps = TC_EstimateSpeed(in);
    TC_ComputeSlip(in, out->v_mps, &out->slip_L, &out->slip_R, &out->slip_avg);

    /* 2. 方向盤濾波 */
    out->steer_deg_f = TC_FilterSteer(in->steer_deg_raw, state);

    /* 3. 目標 slip（模式 + 方向盤） */
    out->lambda_target = TC_ComputeTargetSlip(out->steer_deg_f, out->v_mps, mode);
```

```
    /* 4. 積分重置邏輯 */
    TC_IntegralResetLogic(in, mode, state, out->v_mps);

    /* 5. 自適應 Kp by μ */
    float mu_scale = TC_EstimateMu(state, in);
    float Kp_eff = mode->Kp * mu_scale;
    float Ki_eff = mode->Ki;  // 也可以乘 mu_scale, 看你喜好

    /* 6. PI 計算 */
    float error = out->lambda_target - out->slip_avg;

    float p_term = Kp_eff * error;

    state->integral += error * TC_DT_SEC;
    if (state->integral > INTEGRAL_MAX)   state->integral = INTEGRAL_MAX;
    if (state->integral < -INTEGRAL_MAX)  state->integral = -INTEGRAL_MAX;
    float i_term = Ki_eff * state->integral;

    out->delta_T = p_term + i_term;
}
```

6. Main Loop + Safety + CAN?main.c ???

6.1 ?????TIM2 10 ms ???

```
#include "main.h"
#include "tc_types.h"
#include "tc_mode.h"
#include "tc_fault.h"
#include "tc_controller.h"
#include "tc_io.h"   // 你自己實作: CAN 收發, 按鈕處理

TC_ModeState_t   g_mode;
TC_FaultState_t  g_fault;
```

```
TC_Internal_t    g_state;
TC_Input_t       g_in;
TC_Output_t      g_out;

/* 初始化 */
void TC_SystemInit(void) {
    TC_Mode_Init(&g_mode);
    TC_Fault_Init(&g_fault);
    TC_Controller_Init(&g_state);
}

/* 10 ms 定時中斷 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {

        /* 1. 讀取 CAN / ADC → 填 g_in */
        TC_IO_ReadInputs(&g_in);     // 需要你自己實作

        /* 2. 錯誤偵測 */
        TC_Fault_Update(&g_in, &g_fault);

        /* 3. 模式在靜止時套用 */
        float v_kmh_fake = 0.0f;    // 這裡可以直接用上次 g_out.v_mps 轉 km/h
        v_kmh_fake = g_out.v_mps * 3.6f;
        TC_Mode_ApplyIfStopped(&g_mode, v_kmh_fake);

        /* 4. 判斷是否允許跑 TC */
        bool safe = TC_Fault_IsSafeToRun(&g_fault);

        if (!safe) {
            /* 4-1. 發生嚴重錯誤 → 關閉 TC, 扭矩交回駕駛或安全模式 */
            g_out.tc_status = TC_STATUS_FAULT;
            g_out.T_cmd = 0.0f;  // 或者 0.3 * g_in.T_driver, 看你的需求
        } else {
            /* 4-2. 正常執行控制（含 PI, 但不含最外層 Safety） */
            TC_Controller_Update(&g_in, &g_mode, &g_state, &g_out);

            float v_kmh = g_out.v_mps * 3.6f;

            /* 5. 最外層 Safety 覆蓋 */
```

```
            if (v_kmh < V_TC_OFF_KMH) {
                g_out.T_cmd = g_in.T_driver;
                g_out.tc_status = TC_STATUS_OFF;
            }
            else if (fabsf(g_out.slip_L) > SLIP_SINGLE_MAX ||
                     fabsf(g_out.slip_R) > SLIP_SINGLE_MAX) {
                g_out.T_cmd = 0.5f * g_in.T_driver;
                g_out.tc_status = TC_STATUS_SAFETY;
            }
            else if (fabsf(g_out.steer_deg_f) > 30.0f) {
                g_out.T_cmd = g_in.T_driver * g_mode.min_tq_ratio;
                g_out.tc_status = TC_STATUS_SAFETY;
            }
            else {
                /* □□ PI □□ + □□ */
                float T_raw = g_in.T_driver * (1.0f + g_out.delta_T);
                float T_min = g_in.T_driver * g_mode.min_tq_ratio;
                float T_max = g_in.T_driver;

                if (T_raw < T_min) T_raw = T_min;
                if (T_raw > T_max) T_raw = T_max;

                g_out.T_cmd = T_raw;
                g_out.tc_status = TC_STATUS_NORMAL;
            }
        }

        g_out.fault_flags = g_fault.flags;

        /* 6. □□ CAN □□□T_cmd, □□, □□, fault */
        TC_IO_SendOutputs(&g_in, &g_out, &g_mode, &g_fault);
    }
}
```

7. ???????????tc_io.h / tc_io.c ???

□□□□

- 用 GPIO 按鈕改 `g_mode.target_mode`:
  - 短按 MODE: `STRAIGHT → FIGURE8 → TRACK → STRAIGHT...`
  - 呼叫 `TC_Mode_RequestChange(&g_mode, new_mode, v_kmh)`。
- 由 CAN 顯示:
  - 目前模式 `g_mode.current_mode`。
  - `g_out.tc_status`（OFF / NORMAL / SAFETY / FAULT）。
  - 扭矩限縮比例 `g_out.T_cmd / g_in.T_driver`。
  - µ 等級（可以從 `g_state.ax_peak` 推估）。

你可以自行定義 `0x200 ~ 0x20F` 做儀表板訊息。

---

8. ??????????

1. 先鎖在「繞 **8** 模式」，在乾地繞 8 練習，調整:
   - `LAMBDA_BASE_F8` 先從 0.16 → 看實際 slip 波形。
   - `KP_FIGURE8` 3.1 若太慢可到 3.5；若抖就往下。
2. 直線模式：在直線起步測 `LAMBDA_BASE_STRAIGHT` 0.20 是否太滑 / 太鈍。
3. 賽道模式：上賽道後，再微調 `LAMBDA_BASE_TRACK` 和 `STEER_SENS_TRACK`，讓大轉角出彎更穩。

---

9. ??????

- 模式：決定「基準 λ*、Kp/Ki、最低扭矩比例」。
- 方向盤：決定「當下允許的縱向抓地比例 (k_{steer})」。
- **µ** 估測：決定「Kp 要多兇」。
- 積分重置：保證起步、收油、模式切換時不會被舊積分害死。
- **Safety**: 永遠在最外層，隨時可以一刀切斷 PI 的輸出。

照這份 `.md` 建專案與檔案，功能會跟我們前面討論的一樣齊全，你可以先把這整段貼進 HackMD，之後要補圖（流程圖、波形）可以在各章下面加上自己的實測截圖。

Revision #2
Created 2026-04-01 02:06:25 UTC by TaipeiTechRacing
Updated 2026-04-06 16:39:15 UTC