PLC 梯形圖程式設計審查報告

單車道雙向通行控制（車道感測器 A/B 狀態機）

審查對象： 梯形圖程式（Rung 1–25，疑為 CCW / Micro800 平台）

審查範圍： 邏輯正確性、強健性、安全性、可維護性

報告日期： 2026 年 7 月 5 日

結論摘要： 發現 5 項嚴重問題、6 項重要問題、4 項改善建議

本報告依據五張程式截圖重建邏輯後進行分析；建議修改前先與原始專案檔核對。

Contents

1 程式邏輯重建

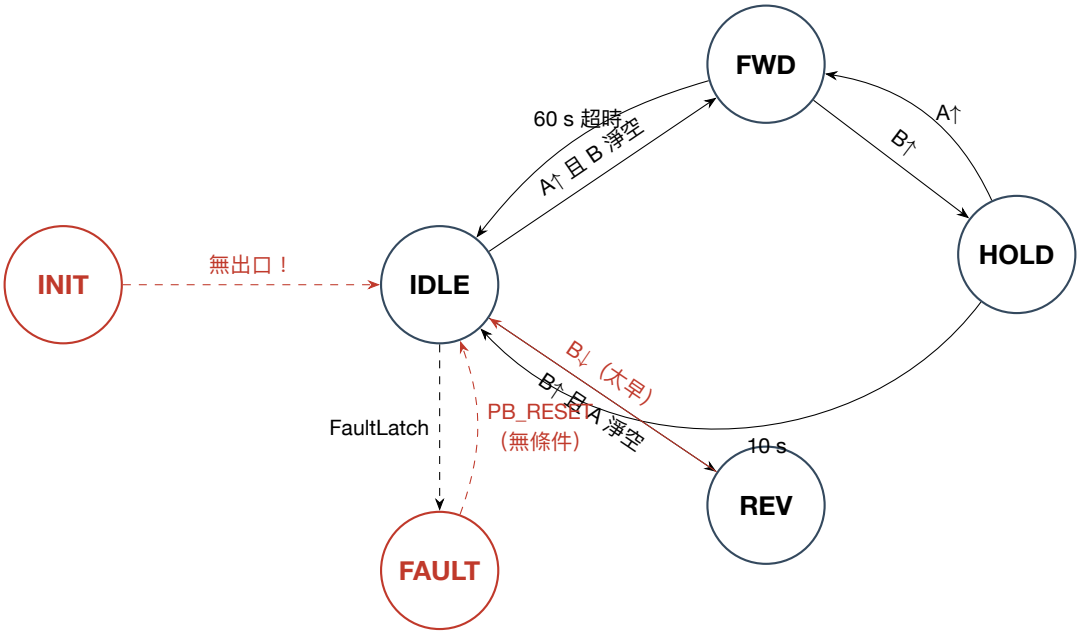
依截圖重建之程式行為如下。系統為**單車道雙向通行控制**：車道兩端各有一支光電感測器（INPUT1 = A 端、INPUT2 = B 端），以整數 STATE 實作狀態機：

值	狀態	意義	備註
0	INIT	上電初始	無任何 rung 處理此狀態
1	IDLE	車道淨空待機	
2	FWD	車輛由 A 往 B 通行中	驅動 OUTPUT_1、60 秒超時
3	HOLD	車輛已過 B，保持 10 秒	驅動 OUTPUT_1
4	REV	車輛由 B 往 A 通行中	不驅動任何輸出、無超時
5	FAULT	感測器卡住故障	FaultLatch 驅動 OUTPUT_2

1.1 Rung 對照表

Rung	功能	邏輯
1–3	INPUT1 邊緣偵測	手寫上升緣 M_INPUT1_Rise、下降緣 M_INPUT1_Fall、上週期值 M_INPUT1_Last
4–6	INPUT2 邊緣偵測	同上，產生 M_INPUT2_*
8	IDLE → FWD	STATE=IDLE 且 A 上升緣且 B 未遮斷
9	IDLE → REV	STATE=IDLE 且 B 上升緣且 A 未遮斷
10	FWD → HOLD	STATE=FWD 且 B 上升緣
11	FWD 超時計時	STATE=FWD 啟動 TON_3（60 s）
12	超時 → IDLE	TON_3.Q 直接 MOV IDLE（未限定狀態）
13	HOLD 計時	STATE=HOLD 啟動 TON_1（10 s）
14	HOLD → IDLE	TON_1.Q 直接 MOV IDLE（未限定狀態）
15	HOLD → FWD	STATE=HOLD 且 A 上升緣（後車跟進）
16	REV → IDLE	STATE=REV 且 B **下降緣**（條件錯誤）
17	車道占用指示	STATE=FWD 或 HOLD → OUTPUT_1
19	卡住偵測計時	INPUT1 或 INPUT2 遮斷即累計 TON_2（5 min）
20–21	故障閂鎖	TON_2.Q → SensorStuck → FaultLatch（L）
22	進入 FAULT	FaultLatch 每掃描週期 MOV FAULT
23	故障復歸	PB_RESET 且 SensorStuck 已消失 → 解閂（U）
24	回到 IDLE	PB_RESET 直接 MOV IDLE（未限定狀態）
25	故障指示	FaultLatch → OUTPUT_2

1.2 現況狀態轉移圖



2 問題分析

2.1 嚴重問題（會造成錯誤動作或死鎖）

問題 1　INIT 狀態沒有出口——上電後永久死鎖

`STATE` 上電為 0（`INIT`），但全程式沒有任何 rung 以 `INIT` 為條件或將其轉移到 `IDLE`。除非變數為保持型（retentive）且曾被手動改寫，否則**每次上電後狀態機完全不動作**，唯一能離開 `INIT` 的路徑竟是按下 `PB_RESET`（Rung 24 的副作用）。

改善：加入首次掃描（first-scan / `_SYSVA_FIRST_SCAN`）rung：上電時檢查兩支感測器皆淨空後 `MOV IDLE → STATE`；若感測器有遮斷則進入 `FAULT` 或等待淨空。

問題 2　Rung 24：PB_RESET 無條件強制回 IDLE

Rung 24 只要按下 `PB_RESET` 就執行 `MOV IDLE → STATE`，完全不檢查目前狀態：

- 車輛正在通行（`FWD`/`REV`/`HOLD`）時誤按復歸，車道占用指示 `OUTPUT_1` 立即熄滅，對向車輛可能進入——**安全隱患**。
- 故障未排除時（`SensorStuck` 仍為真）按住 `PB_RESET`：Rung 22 每週期寫入 `FAULT`，Rung 24 又寫入 `IDLE`，`STATE` 在兩值之間**每週期震盪**；而 Rung 8–17 位於 Rung 22 之前，看到的是上一週期殘留值，行為不可預測。

改善：Rung 24 的條件應為 `PB_RESET` 上升緣且 `STATE = FAULT` 且 `NOT FaultLatch`（即 Rung 23 已成功解閂）。

問題 3　Rung 16：REV 的離開條件錯誤（B 下降緣太早）

車輛由 B 端進入往 A 行駛：遮斷 B → REV；車尾離開 B 感測器時 INPUT2 下降緣成立，狀態**立刻**回 IDLE ——但此時車輛還在車道中間！接著車輛抵達 A 端遮斷 INPUT1，Rung 8 條件（IDLE 且 A↑ 且 B 淨空）全部成立，系統誤判為**一台新車要從 A 進入**，錯誤進入 FWD 並啟動 60 秒計時。單一車輛正常通過就會觸發整條錯誤流程。

改善：REV 應在**車輛通過 A 感測器**（M_INPUT1_Rise 或 M_INPUT1_Fall）時才結束，並比照 FWD 增加 HOLD 緩衝與 60 秒超時。

問題 4　Rung 12/14：計時器 Done 位元未限定狀態

TON_3.Q → MOV IDLE 與 TON_1.Q → MOV IDLE 都沒有串接 EQU(STATE, FWD/HOLD)。目前僅靠「離開狀態後 TON 輸入變假、Q 自動復歸」的副作用維持正確，這依賴掃描順序而且非常脆弱：任何人日後搬動 rung 順序、把計時器改為保持型（RTO）、或在 FAULT 進入的同一週期 Q 尚未復歸，都會讓 IDLE 覆寫掉不該覆寫的狀態（例如把 FAULT 蓋回 IDLE）。

改善：所有寫入 STATE 的 rung 一律以「目前狀態」為第一個條件，例如 EQU(STATE,FWD) AND TON_3.Q → MOV IDLE。

問題 5　疑似標籤不一致：INPUT1_Last 與 M_INPUT1_Last

Rung 1–2 的接點顯示為 INPUT1_Last，但 Rung 3 的線圈寫入的是 M_INPUT1_Last（INPUT2 同樣情形）。若這是兩個不同的變數，**邊緣偵測永遠不會正確動作**（_Last 永遠為 0，上升緣變成電平觸發、下降緣永遠不觸發），整個狀態機行為全部走樣。請開啟專案檔確認是否只是畫面截斷；若真為兩個變數，此為第一優先修復項目。

2.2　重要問題（功能缺陷與安全疑慮）

問題 6　FWD 60 秒超時直接回 IDLE，車輛可能仍在車道內

Rung 12 的設計意圖是處理「車輛迴轉（U-turn）沒走完全程」，但車輛**拋錨卡在車道中間**超過 60 秒時同樣成立：狀態回 IDLE、OUTPUT_1 熄滅，對向車輛會被放行進入仍被占用的車道。**超時屬於異常，應轉入 FAULT（或至少獨立的 TIMEOUT 警告狀態）並保持占用指示**，由人員確認車道淨空後復歸。

問題 7　REV 狀態不驅動任何輸出

Rung 17 只有 FWD/HOLD 會點亮 OUTPUT_1。車輛由 B 往 A 通行期間（REV）車道明明被占用，占用指示卻是熄滅的。若 OUTPUT_1 是「車道占用 / 禁止進入」燈號，這是雙向控制的核心目的，屬於明顯遺漏。**改善：**Rung 17 增加 EQU(STATE,REV) 分支（以及 REV 側若有對應的 HOLD 狀態）。

問題 8　兩端同時來車 → 雙方永久等待

IDLE 時 A、B 幾乎同時遮斷：Rung 8 因 INPUT2 已遮斷不成立，Rung 9 因 INPUT1 已遮斷不成立，兩邊都不轉移。且上升緣旗標只存在一個掃描週期，之後除非車輛倒退重新觸發，否則**兩台車會永遠停在原地等待**。**改善：**改用電平（車輛仍遮斷感測器）搭配先到優先或固

定優先權裁決；或增加「兩側同時佔用」的仲裁狀態。

問題 9　Rung 19：兩支感測器共用一顆卡住偵測計時器

`INPUT1 OR INPUT2` 並聯餵入同一顆 `TON_2`：若 A 端遮斷 4 分鐘後車輛離開、B 端隨即有車遮斷，OR 結果從未斷開，計時**跨感測器連續累積**，1 分鐘後就誤報 `SensorStuck`。此外，合法的長時間停留（如車輛在感測器前排隊等待）也會被判為故障。**改善：**每支感測器各用一顆 TON 獨立計時；並考慮只在「與目前狀態矛盾」時（例如 `IDLE` 卻長時間遮斷）才計時。

問題 10　進入 FAULT 不會即時切斷動作，且復歸順序耦合

Rung 22 位於 Rung 8–17 之後：`FaultLatch` 成立的那個週期，前面的狀態機與輸出仍以舊狀態運作一整個掃描週期。另外復歸流程分散在 Rung 22/23/24 三處且互相依賴掃描順序（先解閂、再回 IDLE、但 Rung 22 又持續覆寫），維護時極易改壞。**改善：**故障偵測 rung 移到狀態機**之前**；復歸整合為單一 rung：`PB_RESET↑ AND STATE=FAULT AND NOT SensorStuck` 時同時解閂並 `MOV IDLE`。

問題 11　多車跟進情境不完整

`FWD` 進行中第二台車在 A 端觸發上升緣會被忽略（沒有任何 rung 消化它），之後首車過 B 進入 `HOLD`，第二台車若已壓在 A 感測器上不再產生新的上升緣，Rung 15 不會成立。**改善：**若場域允許多車，應加入車輛計數（A 進 +1、B 出 -1，REV 反向），以「車道內車輛數 = 0」作為回 `IDLE` 的條件，遠比純計時可靠。

2.3　可維護性建議

建議 12　使用內建邊緣指令取代手寫三段式邊緣偵測

Rung 1–6 共六個 rung 手寫上升/下降緣，等效於平台內建的 `R_TRIG/F_TRIG`（或 OSR/ONS 指令）。手寫版本多兩個中間變數且依賴 rung 順序（`_Last` 必須最後更新），是問題 5 這類錯誤的溫床。

建議 13　狀態機集中化：改用 ST 的 CASE 或單一寫入點

`STATE` 目前有 **8 個分散的寫入點**（Rung 8/9/10/12/14/15/16/22/24），任兩個 rung 在同一週期同時成立時，「最後掃描到的贏」，正確性完全綁死在 rung 排列順序上。建議改用結構化文字（ST）的 `CASE STATE OF` 集中管理（見第 3.2 節範例），或至少保證每個狀態的轉移條件互斥。

建議 14　計時器與變數命名語義化

`TON_1/TON_2/TON_3` 無法從名稱看出用途，建議改為 `tHoldDwell`、`tSensorStuck`、`tFwdTimeout`；`OUTPUT_1/OUTPUT_2` 亦建議改為 `LaneOccupiedLamp`、`FaultLamp`。另建議把 60 s / 10 s / 5 min 等時間常數集中為具名常數，便於現場調整。

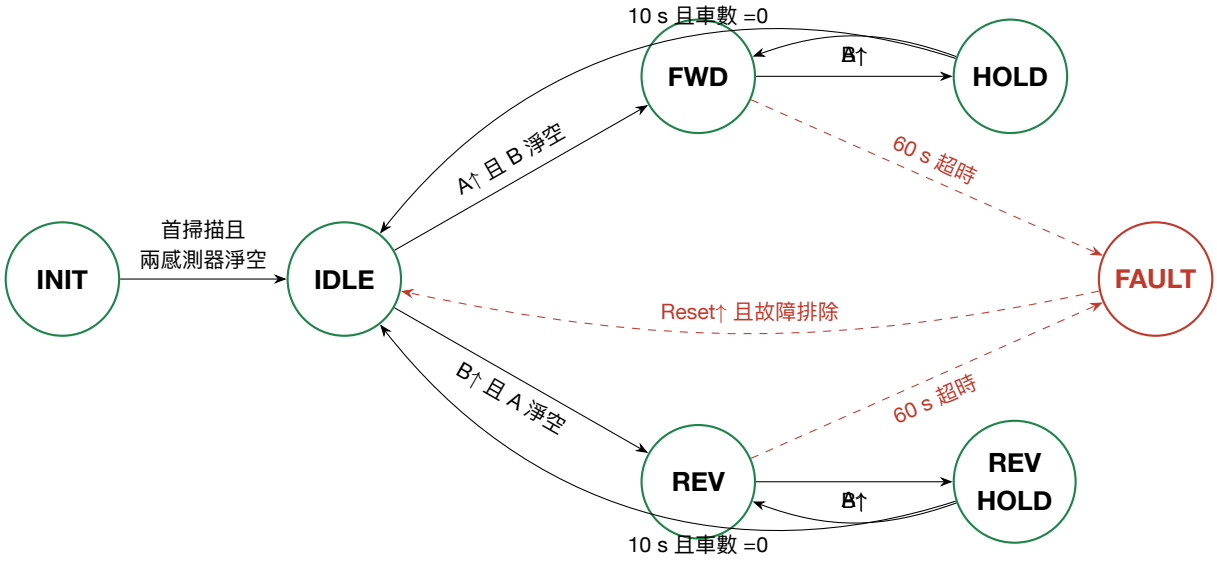
建議 15　PB_RESET 加上升緣處理

Rung 23/24 直接使用按鈕電平。按住不放時 Rung 24 每個週期都執行 `MOV`；搭配問題 2 的無條件寫入，行為更難預測。所有手動按鈕一律建議取上升緣（one-shot）觸發。

3　建議的改善方案

3.1　修正後的狀態機

新增 `REV_HOLD`（B→A 方向的緩衝）與把超時導向 `FAULT`，所有轉移一律以「目前狀態」為前提條件：



目前狀態	事件	次狀態	動作
INIT	首次掃描且 A、B 淨空	IDLE	—
IDLE	A↑ 且 B 淨空	FWD	占用燈 ON、車數 ← 1、啟動 60 s
IDLE	B↑ 且 A 淨空	REV	占用燈 ON、車數 ← 1、啟動 60 s
IDLE	A、B 同時佔用	FAULT/仲裁	依現場規則裁決
FWD	B↑	HOLD	車數 -1、啟動 10 s
FWD	A↑（後車）	FWD	車數 $+1$、重啟 60 s
FWD	60 s 逾時	FAULT	占用燈保持、故障燈 ON
HOLD	10 s 到且車數 $= 0$	IDLE	占用燈 OFF
HOLD	A↑	FWD	車數 $+1$
REV / REV_HOLD	（與 FWD/HOLD 鏡像對稱）	—	—
FAULT	Reset↑ 且感測器已淨空	IDLE	解閂、燈復歸

3.2　結構化文字（ST）重構範例

以 `CASE` 集中所有狀態轉移，`STATE` 只有一個寫入區塊，掃描順序不再影響正確性：

```
(* Edge detection: use built-in blocks instead of hand-written rungs *)
trigA_R(CLK := INPUT1); trigA_F(CLK := INPUT1); // R_TRIG / F_TRIG
```

```
trigB_R(CLK := INPUT2); trigB_F(CLK := INPUT2);

(* Per-sensor stuck detection, only when it contradicts the state *)
tStuckA(IN := INPUT1 AND (STATE = ST_IDLE), PT := T#5m);
tStuckB(IN := INPUT2 AND (STATE = ST_IDLE), PT := T#5m);
IF tStuckA.Q OR tStuckB.Q THEN FaultLatch := TRUE; END_IF;

(* Fault has absolute priority, evaluated BEFORE the state machine *)
IF FaultLatch THEN
   STATE := ST_FAULT;
END_IF;

CASE STATE OF
  ST_INIT:
    IF NOT INPUT1 AND NOT INPUT2 THEN STATE := ST_IDLE; END_IF;

  ST_IDLE:
    IF trigA_R.Q AND NOT INPUT2 THEN
       STATE := ST_FWD; VehCount := 1;
    ELSIF trigB_R.Q AND NOT INPUT1 THEN
       STATE := ST_REV; VehCount := 1;
    END_IF;

  ST_FWD:
    IF trigA_R.Q THEN VehCount := VehCount + 1; END_IF;
    IF trigB_R.Q THEN
       VehCount := VehCount - 1;
       IF VehCount <= 0 THEN STATE := ST_HOLD; END_IF;
    END_IF;
    IF tTransit.Q THEN     // 60 s timeout -> FAULT, not IDLE
       FaultLatch := TRUE; STATE := ST_FAULT;
    END_IF;

  ST_HOLD:
    IF trigA_R.Q THEN STATE := ST_FWD; VehCount := 1; END_IF;
    IF tDwell.Q THEN STATE := ST_IDLE; END_IF;

  ST_REV: (* mirror of ST_FWD, exit on sensor A, own 60 s timeout *)
    // ...

  ST_FAULT:
    IF trigReset.Q AND NOT INPUT1 AND NOT INPUT2 THEN
       FaultLatch := FALSE; STATE := ST_IDLE;
    END_IF;
END_CASE;

(* Timers driven by state *)
tTransit(IN := (STATE = ST_FWD) OR (STATE = ST_REV), PT := T#60s);
tDwell (IN := (STATE = ST_HOLD) OR (STATE = ST_REV_HOLD), PT := T#10s);
```

```
(* Outputs: pure functions of state -- REV now indicates occupancy too *)
LaneOccupiedLamp := (STATE = ST_FWD) OR (STATE = ST_HOLD)
             OR (STATE = ST_REV) OR (STATE = ST_REV_HOLD)
             OR (STATE = ST_FAULT); // keep lane blocked on fault
FaultLamp    := FaultLatch;
```

若必須維持梯形圖，同樣原則可落地為：(1) 故障 rung 移到最前面；(2) 每個 `MOV → STATE` 一律先串 `EQU(STATE, 現態)`；(3) 相同現態的多個轉移條件彼此互斥。

3.3 優先順序總表

優先	項目	一句話摘要
P0	問題 5	確認 `_Last` 標籤是否斷裂，否則邊緣偵測全失效
P0	問題 1	加入首次掃描 INIT→IDLE，否則上電死鎖
P0	問題 3	REV 改以 A 感測器結束，否則單車正常通過即誤判
P0	問題 2	Reset 僅在 FAULT 且故障排除後有效
P1	問題 4	所有 `TON.Q` 轉移補上狀態限定
P1	問題 6、7	超時進 FAULT；REV 也要點占用燈
P2	問題 8、9、11	同時來車仲裁、獨立卡住計時、車輛計數
P3	建議 12–15	R_TRIG/F_TRIG、CASE 集中化、語義化命名、按鈕取邊緣

4 結語

此程式的狀態機骨架（具名狀態常數、邊緣偵測先行、故障閂鎖與獨立復歸條件）方向正確，但目前有四項會在現場直接出事的缺陷：**上電死鎖（INIT 無出口）**、**REV 過早結束導致單車誤判**、**Reset 無條件覆寫狀態**、**以及疑似邊緣偵測標籤斷裂**。建議先以第 3.3 節的 P0 項目止血，再依 P1–P3 逐步把狀態機集中化，讓 `STATE` 只有單一寫入點、輸出成為狀態的純函數，即可大幅降低此類掃描順序相依的隱性錯誤。