COMSM1302 電腦架構概論

Lab Sheet 4 完整詳解

CMOS 電晶體邏輯閘 · 升／降緣偵測器 · 程式計數器 · 紅綠燈有限狀態機

逐題解析（含完整設計過程）

2026 年 7 月

本實驗的學習目標：完成本 Lab 後，你應該能夠——

1. 以電晶體為積木建出邏輯閘；
2. 建出使用循序邏輯的時脈電路；
3. 設計實作有限狀態機（FSM）的電路。

所需工具：Logisim（wiring 函式庫的電晶體、電源、接地；memory 函式庫的暫存器與 D 正反器）。本週往抽象階梯的**兩端**同時延伸：向下鑽到電晶體（閘是怎麼來的），向上蓋出計數器與狀態機（CPU 的雛形）。

目錄

1　電晶體（Transistors）

題目

在 Logisim 上用電晶體建出以下邏輯閘：NOT A、A NAND B、A AND B、A NOR B、A OR B。需使用 1 位元輸入／輸出腳位、p 型電晶體（朝南）、n 型電晶體（朝北）、電源元件（V_{dd}）與接地元件（V_{ss}），皆在 wiring 函式庫中。

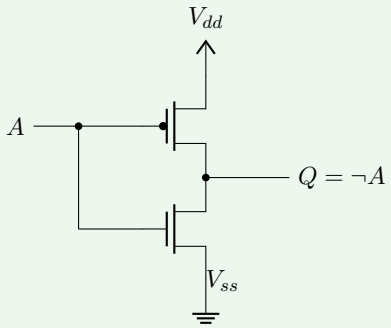
解答

先複習兩種電晶體的開關規則（第四週講義）：

種類	gate= 0	gate= 1	擅長傳遞
n 型（無圓圈）	斷開	**導通**	0（放在 V_{ss} 側，拉低）
p 型（gate 有圓圈）	**導通**	斷開	1（放在 V_{dd} 側，拉高）

CMOS 的設計配方：每個閘 = **上拉網路**（p 型，接 V_{dd}，負責輸出 1）+ **下拉網路**（n 型，接 V_{ss}，負責輸出 0），兩個網路互補——任何輸入組合下**恰好一邊導通**，輸出永遠被明確驅動，且沒有從 V_{dd} 直通 V_{ss} 的漏電路徑。

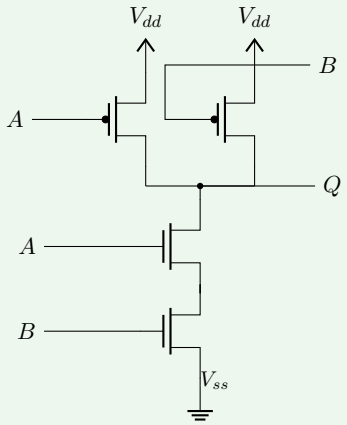
(1) NOT A（2 顆）。



$A = 0$：p 導通、n 斷開 ⇒ Q 被拉到 V_{dd}（1）；$A = 1$：p 斷開、n 導通 ⇒ Q 被拉到 V_{ss}（0）✓。

(2) A NAND B（4 顆）：p 並聯、n 串聯。

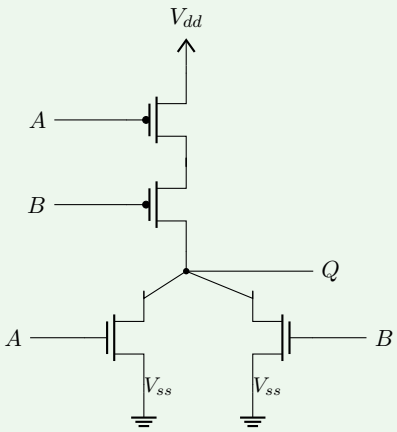
輸出為 0 的唯一條件是 $A = B = 1$ ⇒ 下拉路徑必須「兩個都導通才通」——**串聯**；輸出為 1 的條件是「任一為 0」⇒ 上拉路徑「任一導通就通」——**並聯**。



逐列驗證：A=B=0：兩 p 皆通 → Q=1；A=0, B=1：p_A 通 → Q=1；A=1, B=0：p_B 通 → Q=1；A=B=1：兩 n 皆通 → Q=0 ✓（正是 NAND）。

(3) A NOR B（4 顆）：p 串聯、n 並聯。

與 NAND 完全對偶：輸出為 1 的唯一條件是 $A = B = 0 \Rightarrow$ 上拉**串聯**；任一為 1 就輸出 0 ⇒ 下拉**並聯**。



(4) A AND B（6 顆）= NAND + NOT。

CMOS 天生只會做「反相」閘（p 在上拉、n 在下拉的結構必然輸出反函數）。要做 AND，把 (2) 的輸出接到 (1) 的輸入：$A \wedge B = \neg(A \barwedge B)$。共 $4 + 2 = 6$ 顆。

(5) A OR B（6 顆）= NOR + NOT。

同理：$A \vee B = \neg(A \downarrow B)$，把 (3) 的輸出接反相器。共 $4 + 2 = 6$ 顆。

Logisim 組裝細節（wiring 函式庫）：

- **p 型朝南（facing south）**：訊號從上方的 V_{dd} 流向下方輸出——p 型負責「往下推 1」；**n 型朝北（facing north）**：訊號從下方的 V_{ss} 流向上方輸出——n 型負責「往上拉 0」。箭頭方向就是電流容許的流向，接反了值傳不過去；
- gate 腳位在側面（屬性 Gate Location 可選左右），p 型的 gate 有小圓圈（「0 才動作」的記號，與主動低的氣泡是同一語言）；
- Logisim 的電晶體是**數位近似**：不導通時輸出 **浮接（Z）**；若浮接值再穿過另一顆導通的電晶體，某些版本會變成**錯誤值（紅線）**——這發生在串聯堆疊的**中間節點**（例如 NAND 的 A=1, B=0 時）。遇到紅線可在中間節點加一顆 pull resistor （wiring 函式庫）救援，或改用官方 Logisim 2.7；
- 逐閘驗證：接 1 位元輸入腳位、用手（poke 工具）跑完整張真值表再往下一題。

進階討論

為什麼 AND 要 6 顆而 NAND 只要 4 顆? 這正是第四週講義「NAND 天生比較快」的電路層原因：CMOS 結構中 p 管上拉、n 管下拉的分工使每個基本閘必然是**反相**的（輸入 1 導致下拉、輸出變 0）。「正相」閘（AND、OR）都得再付一級反相器的代價——所以工程師寧可用 NAND/NOR 直接設計，或用「氣泡推移」把反相吸收進相鄰的閘。另外注意 NAND 與 NOR

的對偶：NAND 是「n 串聯」、NOR 是「p 串聯」——而 p 管靠電洞導電、遷移率只有電子的約一半，串聯兩顆慢上加慢，這就是 NOR 比 NAND 慢、晶片世界偏愛 NAND 的物理根源。

2　升／降緣偵測器（Rise/Fall Detector）

題目

用課堂提供的 one-shot 電路設計一個升／降緣偵測器：時脈電路，1 位元輸入 `in`、兩個 1 位元輸出 `rise` 與 `fall`。在時脈上升緣：

- 若 `in` 自上個上升緣以來由低變高，`rise` 應輸出一個時脈週期的高電位脈衝；
- 若 `in` 由高變低，`fall` 應輸出一個週期的脈衝。

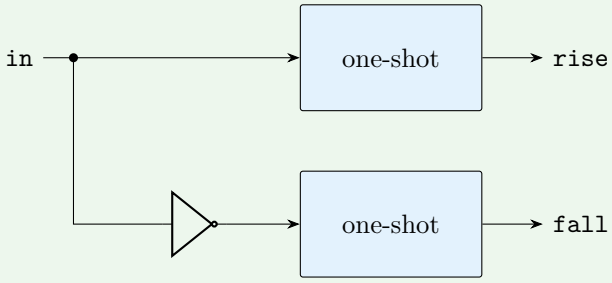
先用現成的 one-shot 當子電路建；再改造 one-shot 的 Mealy 機設計直接建。

解答

先回顧課堂的 one-shot。 one-shot 把「長按的 1」變成「恰好一拍的 1」。講義給了兩個版本：

- **Moore 版**（3 狀態、2 個正反器）：$X_{new} = (X \vee Y) \wedge in$、$Y_{new} = \neg X \wedge \neg Y \wedge in$、$out = Y$；
- **Mealy 版**（1 個正反器）：用正反器**把輸入本身存一拍**，$S_{new} = in$，$out = \neg S \wedge in$ ——「現在是 1、上一拍是 0」＝**剛升起**。

做法一：用 one-shot 子電路組裝（2 個 one-shot ＋ 1 個 NOT）。
關鍵觀察：`in` 的**下降**緣就是 `¬in` 的**上升**緣。所以：



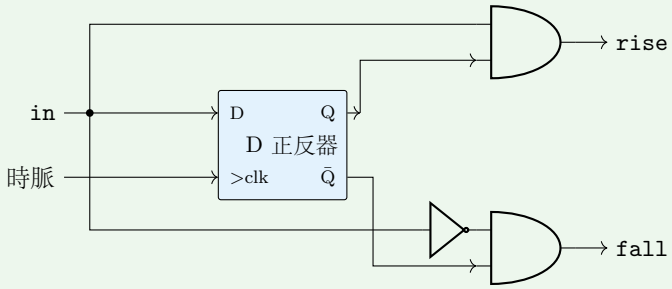
（兩個子電路共用同一條時脈）

`rise` = one-shot(`in`)：`in` 升起後脈衝一拍 ✓；`fall` = one-shot(`¬in`)：`in` 降下（`¬in` 升起）後脈衝一拍 ✓。在 Logisim 中把課堂下載的 one-shot 電路存成 subcircuit，放兩份即可。

做法二：直接改造 Mealy 設計（1 個正反器＋ 2 個 AND ＋ 1 個 NOT）。
Mealy one-shot 的正反器存的是「上一拍的輸入」S。比較 S（過去）與 `in`（現在）就能同時偵測兩種邊緣：

S（上一拍）	`in`（現在）	`rise`	`fall`	情況
0	0	0	0	一直是低
0	1	**1**	0	剛升起
1	0	0	**1**	剛降下
1	1	0	0	一直是高

$$\texttt{rise} = in \wedge \neg S, \qquad \texttt{fall} = \neg in \wedge S.$$



省閘技巧（講義同款）：$\neg S$ 不必用 NOT 閘——D 正反器本來就有 $\bar{Q}$ 輸出。上圖 `rise` 用了 Q ……更精確的接法：$\texttt{rise} = in \wedge \bar{Q}$（接 $\bar{Q}$）、$\texttt{fall} = \neg in \wedge Q$（接 Q，`in` 過一個 NOT）。總成本：1 個正反器、2 個 AND、1 個 NOT。

行為驗證（時脈邊緣編號 $t_0, t_1, \ldots$）：設 `in` 在 t_1 與 t_2 之間升起。t_2 時 S 仍是 0（上一拍存的），`in` 已是 1 ⇒ `rise`= 1；t_2 邊緣後 S 更新為 1 ⇒ t_3 起 `rise`= 0——恰好一拍的脈衝 ✓。下降對稱同理 ✓。

這個小電路是真實硬體的常客：按鍵中斷（偵測按下／放開）、通訊協定的邊緣同步（UART 起始位元偵測）、計數器的事件觸發都用它。它也展示了 Mealy 機的招牌優點：輸出取決於「狀態＋輸入」的**組合**，比 Moore 機（輸出只看狀態）常常省下狀態數——這裡 1 個正反器就同時偵測兩種邊緣，Moore 版本得用 3 個狀態以上才做得到同樣的事。

3　程式計數器（Program Counter）

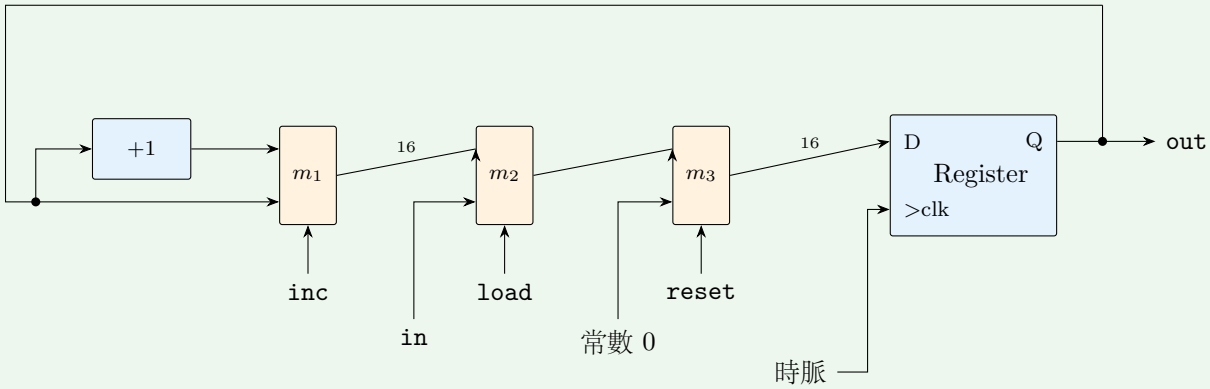
題目

在 Logisim 上實作程式計數器：三個 1 位元輸入 `reset`、`inc`、`load`，一個時脈輸入，16 位元輸入 `in`，16 位元輸出 `out`。在時脈上升緣：`reset` 高 → `out` 設為 0；`load` 高 → `out` 設為 `in`；`inc` 高 → `out` 設為 `out`+1。三者至多一個為高。邊緣之後 `out` 保持不變。

解答

架構：一顆 16 位元暫存器＋「下一個值」的組合邏輯。「邊緣之後保持不變」直接告訴我們核心是**邊緣觸發的暫存器**；三種行為只是決定**下一個值**是什麼——用三顆 16 位元 2-to-1 多工器逐級選擇：

$$next \;=\; \mathrm{MUX}\Big(\mathrm{MUX}\Big(\underbrace{\mathrm{MUX}(out,\, out+1,\, inc)}_{m_1},\, in,\, load\Big),\, 0,\, reset\Big)$$
$$\underbrace{\qquad\qquad\qquad\qquad}_{m_2}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{m_3}$$



Logisim 組裝清單：

- **暫存器**：memory 函式庫 Register，Data Bits = 16，時脈直接接上、enable 恆為 1（每個上升緣都寫入 *next*——三個控制都是 0 時 m_1 選的是 `out` 自己，寫回原值等於「保持」）；
- **加一器**：arithmetic 函式庫的 Adder （一端接常數 1）；若想貫徹第二週精神，可用自己做的 4 位元遞增器擴成 16 位元；
- **三顆 MUX**：plexers 函式庫，Data Bits = 16、Select Bits = 1；選擇端分別接 `inc`、`load`、`reset`；
- **常數 0**：wiring 函式庫 Constant （Data Bits = 16、值 `0x0000`）。

為什麼把 `reset` 放在最後一級？ 離暫存器最近的 MUX 優先權最高。題目雖保證三者至多一個為高，但把優先序排成 *reset* > *load* > *inc* 是業界慣例（重開機必須壓倒一切）——這恰好是 nand2tetris 的 PC 晶片規格：

```
if reset: out=0   elif load: out=in   elif inc: out=out+1   else: out=out
```

驗證流程： `reset` 打一拍 → `out`= 0；`inc`= 1 連走五拍 → 0,1,2,3,4,5；`load` 配 `in`= 100 打一拍 → 100；再 `inc` 兩拍 → 101, 102；全部拉 0 走幾拍 → 維持 102 ✓。

它為什麼叫「程式計數器」？ 這顆元件就是第五週 Hack CPU 的 `PC`：平常 `inc`（逐條執行下一條指令）、跳躍時 `load`（把 A 暫存器的目標位址載入）、開機時 `reset`（從 ROM[0] 開始）。你剛剛做完了 CPU 的「導航系統」。

4　紅綠燈（Traffic Lights）

行人按下按鈕後，紅綠燈依序循環：綠 ⇒ 黃 ⇒ 紅 ⇒ 閃黃 ⇒ 綠。使用 **one-hot 編碼**（每個狀態一個位元、任何時刻恰好一個位元為 1）：

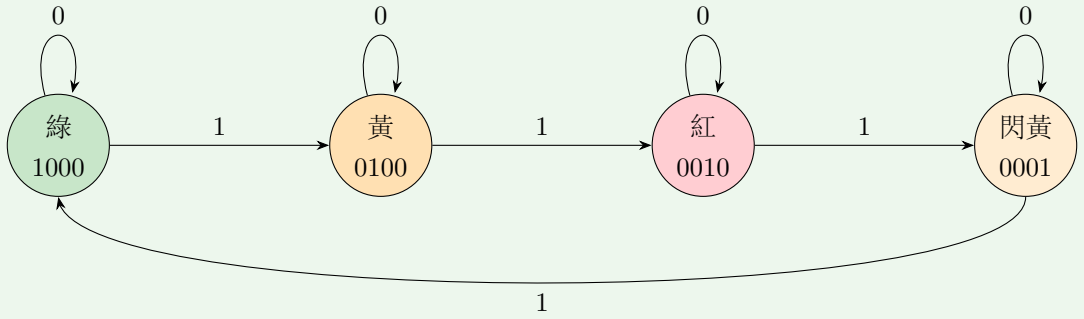
狀態	編碼 $GARF$
綠（green）	1000
黃（amber）	0100
紅（red）	0010
閃黃（flashing amber）	0001

4.1　狀態轉移圖——Moore 還是 Mealy?

題目

畫出狀態轉移圖，轉移由 1 位元輸入 `change` 決定：`change` 高則進到序列的下一個狀態、低則不動。這是 Moore 還是 Mealy 機?

解答



（弧上的數字是輸入 `change` 的值。）

這是 Moore 機。判準：輸出（亮哪盞燈）**只由目前狀態決定**——狀態位元本身就是輸出，輸入 `change` 只影響**轉移到哪裡**，不直接影響輸出。若輸出同時取決於狀態與當下輸入（像第 2 節 Mealy 版 one-shot 的 $out = \neg S \wedge in$），才是 Mealy 機。

4.2　狀態儲存電路

題目

建一個電路儲存狀態：輸入 `change`（1 位元），輸出四個狀態位元。每個時脈週期依轉移圖更新。開機後（至少幾拍之內）狀態應為綠。提示：one-shot 可用來設定初始狀態。

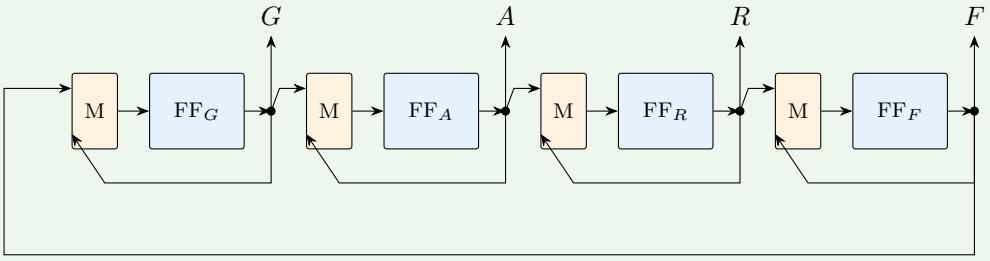
解答

one-hot 編碼讓下一狀態邏輯出奇地簡單。「進到下一個狀態」= 把那顆 1 **往右輪轉一格**（F 之

後繞回 G）；「不動」= 各位元保持。每個位元的方程式：

$$G_{new} = (change \wedge F) \vee (\neg change \wedge G),$$
$$A_{new} = (change \wedge G) \vee (\neg change \wedge A),$$
$$R_{new} = (change \wedge A) \vee (\neg change \wedge R),$$
$$F_{new} = (change \wedge R) \vee (\neg change \wedge F).$$

硬體上這是一個**可控輪轉的環形移位暫存器**：4 顆 D 正反器排成環，每顆 D 輸入前放一顆 2-to-1 MUX （select = `change`；0 端接自己的 Q、1 端接**前一顆**的 Q）。



（F 繞回 G；四顆 MUX 的 select 全接 `change`，四顆 FF 共用時脈）

開機初始化——one-shot 登場。 Logisim 的正反器開機全為 0，狀態 0000 不合法（沒有任何燈亮，而且 0 在環裡轉永遠是 0000）。解法：把**常數 1** 接進課堂 one-shot——開機後它會輸出**恰好一拍**的脈衝 $init$；用 OR 閘把 $init$ 注入綠位元：

$$G_{new} = init \vee \big[(change \wedge F) \vee (\neg change \wedge G)\big].$$

第一拍過後 $G = 1$、其餘為 0，狀態機從綠燈正常起跑（題目允許「開機後前幾拍內」就位）✓。**更穩健的替代方案**（非必要但值得知道）：$init = \neg(G \vee A \vee R \vee F)$——「四個位元全滅」本身就是非法訊號，用一顆 NOR 偵測並注入 G。好處是電路**自我修復**：即使日後因雜訊掉進非法狀態，下一拍就會回到綠燈。

4.3　計時電路

題目

建一個計時電路：輸入 `start`（1 位元）、`time`（16 位元），輸出 `done`（1 位元）。`start` 由 0 變 1 時，把 `time` 當二進位數，等那麼多個時脈週期，然後讓 `done` 維持高，直到 `start` 變回 0。可假設 `done` 變高之前 `start` 不會先變回 0。提示：可做一個「判斷兩數相等」的子電路。

解答

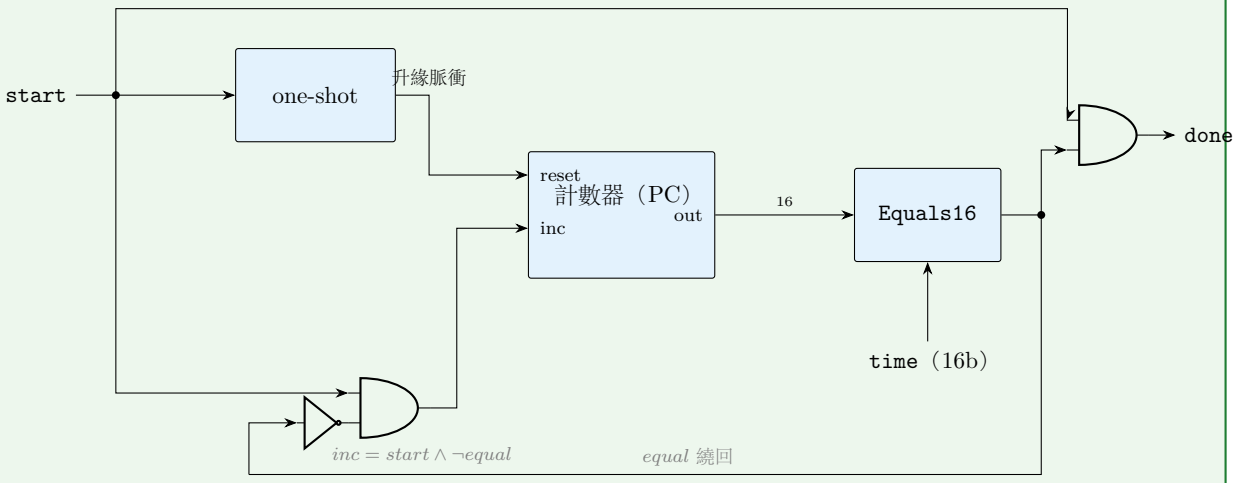
先做提示的相等比較器。兩個位元相等 ⇔ XNOR 為 1；兩個 16 位元數相等 ⇔ **每一對**位元都相等：

$$equal = \bigwedge_{i=0}^{15} \neg(a_i \oplus b_i) \qquad \text{（16 顆 XNOR + AND 樹）}$$

Logisim 實作：兩個 splitter 把 16 位元拆開、16 顆 XNOR、再用多輸入 AND（屬性可調到 16

輸入，或蓋一棵兩兩 AND 的樹）。存成子電路 `Equals16`。

計時器主體 ＝ 計數器＋比較器＋控制。直接復用第 3 節的程式計數器：



三條控制訊號的邏輯：

- **歸零**：`start` 的上升緣（one-shot 產生的一拍脈衝）接計數器的 `reset` ——每次啟動從 0 重新數；
- **計數**：$inc = start \wedge \neg equal$ ——`start` 為高且還沒數到目標就繼續加一；數到之後 *inc* 變 0，計數器**凍結**在 `time`，*equal* 因此持續為真（訊號自我保持，不需要另加閂鎖）；
- **完成**：$done = start \wedge equal$ ——`start` 拉回 0 的瞬間 `done` 跟著歸 0，完全符合「維持到 `start` 變回 0 為止」。

時序驗證：`start` 升起（第 0 拍歸零），第 1 拍起計數 $1, 2, \ldots$，第 `time` 拍計數值到達 `time` ⇒ `done` 升起——恰好等了 `time` 個時脈週期 ✓。

4.4　整合：定時紅綠燈

題目

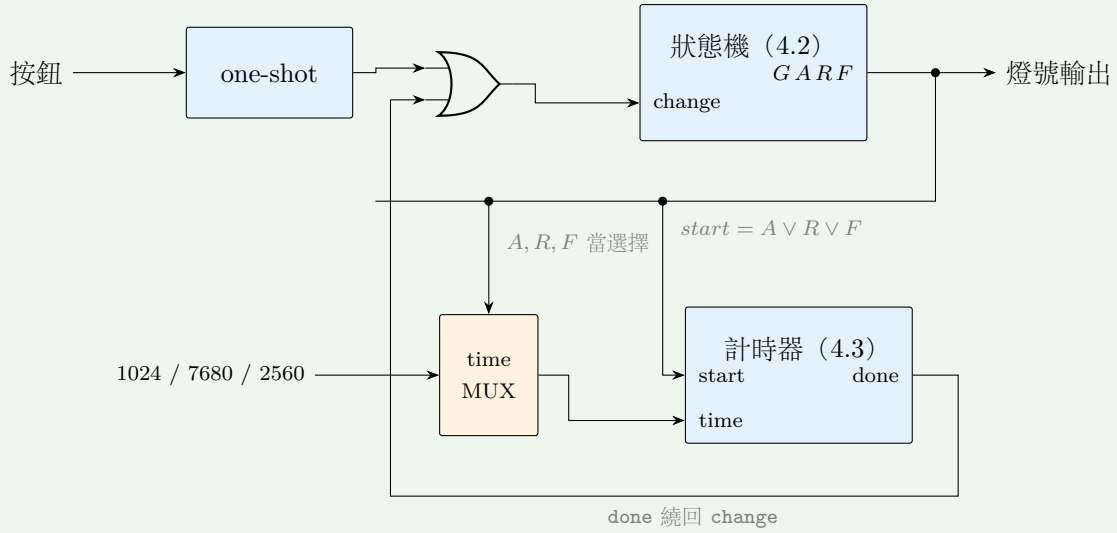
用 4.2 與 4.3 的設計作子電路，建出定時紅綠燈：一個 1 位元輸入（按鈕）啟動以下序列——綠 → 黃（立即）、黃 → 紅（約 2 秒後）、紅 → 閃黃（約 15 秒後）、閃黃 → 綠（約 5 秒後）。時脈用 512Hz；閃黃期間黃燈以約 1.25Hz 在 1 與 0 之間交替、以亮起開始。提示：可做一個產生閃爍行為的子電路。

解答

第一步：把「秒」換算成「拍」。時脈 512Hz ⇒ 每秒 512 拍：

停留狀態	時間	拍數（`time` 值）
黃	2 秒	$2 \times 512 = 1024$
紅	15 秒	$15 \times 512 = 7680$
閃黃	5 秒	$5 \times 512 = 2560$

第二步：頂層架構。兩個子電路的分工：狀態機管「現在哪個燈」、計時器管「什麼時候換」。膠合邏輯只有三件事：



1. **啟動**：按鈕過 one-shot 變成一拍脈衝 *press*（行人按多久都只算一次）。$change = press \vee done$——按下按鈕（綠 → 黃立即）或計時到期（其餘轉移）都推進狀態；

2. **選時間**：*time* 由目前狀態經 MUX 選出——$A \to 1024$、$R \to 7680$、$F \to 2560$（三個 16 位元常數 + 用狀態位元當選擇的 MUX；one-hot 編碼下可直接用兩顆 4-to-1 MUX 的組合，或最簡單：$time = (A{\cdot}1024) \vee (R{\cdot}7680) \vee (F{\cdot}2560)$，每個常數與其狀態位元逐位 AND 再 OR 起來——不會衝突，因為恰好一個狀態位元為 1）；

3. **何時計時**：$start = A \vee R \vee F$（綠燈不計時，等按鈕）。**關鍵細節**：黃 → 紅、紅 → 閃黃的轉移瞬間 *start* 保持為 1 不會有下降緣，計時器的內部 one-shot 偵測不到「重新啟動」——所以計時器的**歸零訊號**要改接頂層的 *change* 脈衝（每次進入新狀態就重數），這是對 4.3 子電路的一行小改（把 reset 輸入外露）。改完後 *done* 每次到期只持續到下一拍（計數器被 *change* 歸零、*equal* 隨即消失），恰好是一拍的 *change* 脈衝——時序自洽 ✓。

第三步：閃爍子電路（提示要求的）。閃爍頻率約 1.25Hz ⇒ 完整亮滅週期 $= 512/1.25 = 409.6$ 拍 ⇒ 每 ≈ 205 拍翻轉一次（$512/(2 \times 205) = 1.2488$Hz ✓「約 1.25Hz」）。組件：

- 一顆小計數器 + `Equals16`（比對 204）：數到 204 就發脈衝 *tick* 並自我歸零（*tick* 接回計數器 reset）——「每 205 拍一個脈衝」；
- 一顆 T 正反器（上週做的!）：由 *tick* 觸發翻轉，輸出 *blink*；

- **以亮起開始**：進入閃黃的那一拍（$change \wedge R$，即「紅燈時發生轉移」）把 T 正反器**設為 1**、小計數器歸零——燈先亮滿 0.4 秒再開始交替。

最後把燈接起來：

$$\text{綠燈} = G, \qquad \text{黃燈} = A \vee (F \wedge blink), \qquad \text{紅燈} = R.$$

整體驗證（Logisim 時脈設 512Hz，Simulate → Auto-Tick）：按一下按鈕 → 黃燈立亮；約 2 秒後轉紅；約 15 秒後黃燈開始閃（先亮）；約 5 秒後回綠、靜候下一位行人 ✓。

進階討論

為什麼用 one-hot 而不是二進位編碼?四個狀態用二進位只要 2 個正反器(00,01,10,11),one-hot 卻用 4 個——看似浪費，但換來：

- **下一狀態邏輯極簡**：輪轉一格 vs. 二進位需要「加一再解碼」的組合邏輯；
- **輸出零成本**：狀態位元就是燈號，不需輸出解碼器（二進位編碼還得配 2-to-4 解碼器）；
- **速度**：每個位元的邏輯只有一顆 MUX 深，時脈可以跑更快——FPGA 工具鏈至今預設偏好 one-hot。

代價是正反器數量與「非法狀態」變多（$2^4 - 4 = 12$ 個非法組合），所以 4.2 的自我修復設計在工程上格外有價值。**對照真實世界**：真的行人紅綠燈控制器正是一顆跑 FSM 的微控制器或 PLC，「按鈕 → one-shot → 狀態機 + 計時器」的骨架與你在本 Lab 蓋的一模一樣。

A 附錄：速查表

A.1 CMOS 閘的電晶體計數

閘	電晶體數	結構
NOT	2	1p + 1n
NAND	4	2p 並聯 + 2n 串聯
NOR	4	2p 串聯 + 2n 並聯
AND	6	NAND + NOT
OR	6	NOR + NOT

A.2 本 Lab 的關鍵方程式

電路	方程式
Mealy one-shot	$S_{new} = in;\ \ out = \neg S \wedge in$
升降緣偵測	$rise = in \wedge \neg S;\ \ fall = \neg in \wedge S$
程式計數器	$next = reset\,?\,0 : load\,?\,in : inc\,?\,out{+}1 : out$
one-hot 輪轉	$S_i^{new} = (change \wedge S_{i-1}) \vee (\neg change \wedge S_i)$
相等比較	$equal = \bigwedge_i \neg(a_i \oplus b_i)$
計時器控制	$inc = start \wedge \neg equal;\ \ done = start \wedge equal$
燈號輸出	黃燈 $= A \vee (F \wedge blink)$

A.3 時間換算（時脈 512Hz）

需求	換算	拍數
黃燈 2 秒	2×512	1024
紅燈 15 秒	15×512	7680
閃黃 5 秒	5×512	2560
閃爍 1.25Hz 半週期	$512/(2 \times 1.25) = 204.8$	≈ 205

A.4 參考資料

- COMSM1302 第四週講義（正反器與暫存器的積木應用、Moore ／ Mealy 機、電晶體與 CMOS），University of Bristol.
- Logisim Library Reference——Transistor （p 型 gate= 0 導通、n 型 gate= 1 導通、不導通時輸出浮接）.
- Nisan & Schocken, *The Elements of Computing Systems*（nand2tetris），Ch. 3——PC（程式計數器）晶片規格.
- Harris & Harris, *Digital Design and Computer Architecture*——CMOS 邏輯、FSM 狀態編碼（binary vs. one-hot）.