COMSM1302 電腦架構概論

第四週完整教材：狀態機與電晶體

計數器與單觸發 · 有限狀態機（Moore ／ Mealy）· 電晶體與 CMOS · NAND vs NOR · SRAM vs DRAM

依據 John Lapinskas 與 Kira Clements（University of Bristol）課程講義編寫並延伸

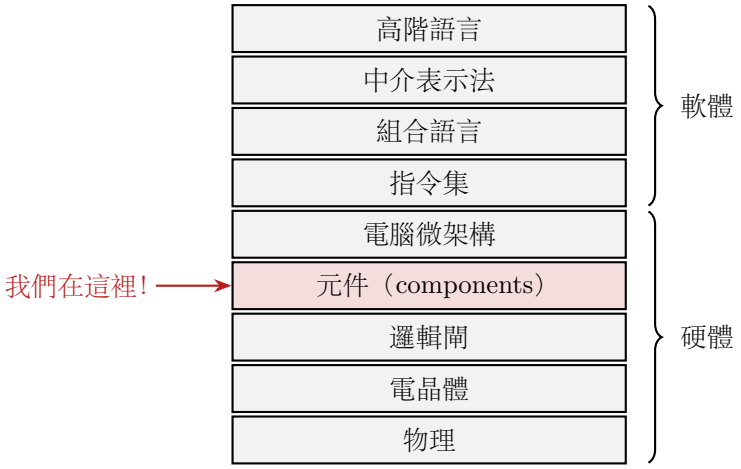
2026 年 7 月

本教材的使用方式：本講義整合了第四週四份投影片（4-1 Building with flip-flops and registers、4-2 Transistors、4-3 NAND vs NOR、4-4 SRAM vs DRAM）的全部內容，並補充了狀態編碼、Moore ／ Mealy 取捨、邏輯努力（logical effort）、6T SRAM 與 1T1C DRAM 等延伸知識。每章結尾附有「本章重點」整理；第 5 章為綜合練習題，附完整詳解。本週在抽象階梯上**同時往上、往下**走：往上——用暫存器組出計數器與**有限狀態機**（通往 CPU 控制邏輯）；往下——打開邏輯閘，看到底層的**電晶體**與 CMOS 電路（回答「NAND 為什麼是工業標準」「SRAM ／ DRAM 裡面長什麼樣」）。

目錄

1 用正反器與暫存器蓋東西（Building with Flip-Flops and Registers）

1.1 從閘到元件：抽象階梯



事情已經複雜到**不能再用一顆顆 NAND 來思考**了！我們必須把東西抽象成**元件（components）／子電路（subcircuits）**——多工器、加法器、暫存器、邏輯閘；只有在 NAND 恰好是對的工具時才用 NAND。而這些元件本身又會成為更大元件的一部分……直到蓋出整台電腦。

1.2 暫存器作為積木：計數器（Counter）

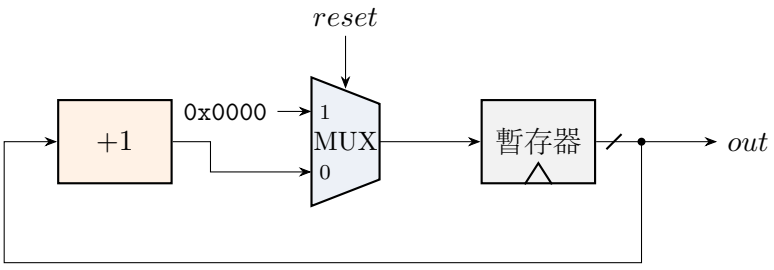
定義

計數器（counter）儲存並輸出一個二進位值 out，在每個時脈（上升緣）**加 1**。它只有一個輸入 $reset$：若上升緣時 $reset = 1$，計數器的值歸零。

怎麼用 16 位元暫存器蓋出 16 位元計數器？沿用上週暫存器的思路——寫一張非正式的「真值表」，用輸入與**上一拍的輸出** out_{old} 表達**想要的下一拍輸出** out_{new}：

$reset$	out_{new}
0	$out_{\text{old}} + 1$
1	0x0000

兩個現成工具：$out_{\text{old}} + 1$ 用**加法器**算（第二週！）；「若 0 則 x、否則 y」的選擇邏輯用**多工器**（第二週！）：



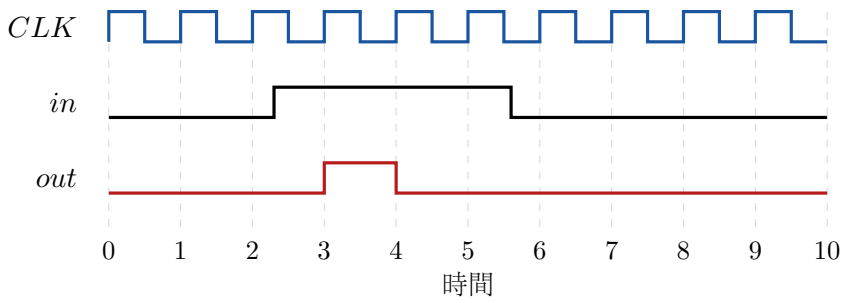
資料流：暫存器輸出 *out* 回饋給「+1」加法器；MUX 依 *reset* 在「$out_{\text{old}}+1$」與常數 0 之間二選一，餵回暫存器（*load* 恆為 1）。每個上升緣自動前進一步。

這個模式極其重要：「暫存器存狀態 → 組合邏輯算下一個狀態 → 餵回暫存器」。上週的暫存器（MUX 選舊值或新值）、本週的計數器、下面的狀態機、乃至 CPU 的程式計數器（program counter），全部都是這一個模式的實例。

1.3 正反器作為積木：單觸發（One-Shot）

定義

單觸發（one-shot）把一段（可能很長的）1 輸入變成**恰好持續一個時脈週期**的 1 脈衝。

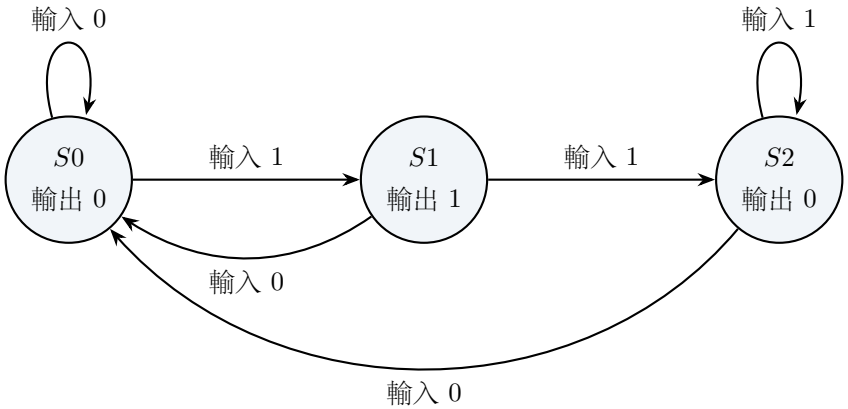


（*in* 在 $t \approx 2.3$ 變高並持續到 $t \approx 5.6$；*out* 只在 $t = 3$ 的上升緣輸出一拍的 1。）用途：開機時初始化暫存器、把使用者的長按按鈕轉成方便處理的單拍訊號等。

怎麼用正反器蓋？訣竅是把電路的行為**拆解成「狀態」**，存進正反器／暫存器。

1.4 狀態圖（State Diagrams）

我們想要單觸發做三件事：(a) 等待 *in* 出現 1；(b) 送出一拍脈衝；(c) 等 *in* 回到 0，再回到 (a)。畫成圖（節點 = 狀態，上為名稱、下為**該狀態的輸出**；箭頭 = 輸入觸發的轉移）：



逐一對照需求：$S0$ =「等待 1」（輸出 0）；看到 1 進入 $S1$ =「送脈衝」（輸出 1，只停留一拍）；若 *in* 仍是 1 進入 $S2$ =「等待 0」（輸出 0，防止重複觸發）；*in* 回 0 才回到 $S0$。

定義

這是一台 **Moore 機（Moore machine）**：每個時脈週期依輸入在狀態間轉移，且**輸出只是狀態的函數**（寫在節點裡）。把狀態 S 用二進位數存在暫存器／正反器裡，狀態轉移與輸出就都變成**組合邏輯**！

1.5 打造單觸發（Moore 版）

用兩個正反器 XY 存狀態：$S0 = 00$、$S1 = 01$、$S2 = 10$。寫出**狀態轉移真值表**與**輸出真值表**：

X_{old}	Y_{old}	in	X_{new}	Y_{new}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	X	X	X

X	Y	out
0	0	0
0	1	1
1	0	0
1	1	X

（狀態 11 不存在，全部填 X——don't care 讓化簡更自由。）讀表化簡：

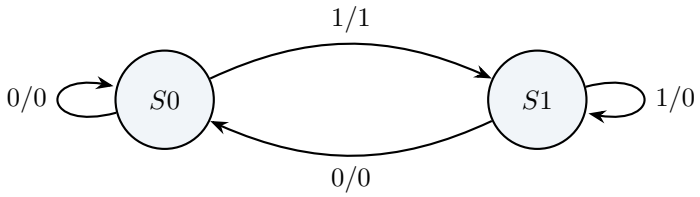
$$X_{\text{new}} = (X_{\text{old}} \vee Y_{\text{old}}) \wedge in, \qquad Y_{\text{new}} = \neg X_{\text{old}} \wedge \neg Y_{\text{old}} \wedge in, \qquad out = Y.$$

逐條驗證：$X_{\text{new}} = 1$ 的兩列（011、101）都滿足「舊狀態非 $S0$ 且 $in = 1$」✓；$Y_{\text{new}} = 1$ 只有一列（001）：「舊狀態是 $S0$ 且 $in = 1$」✓；輸出直接取 Y（利用 don't care 把 11 當 1 也無妨）✓。

1.6 更好的單觸發：Mealy 機

這樣就最佳了嗎？**絕對不是**！有很多技巧可以做得更好，例如非常小心地選擇**狀態編碼**方式。

這裡示範一招：**把輸入本身也存進一個正反器**，讓輸出可以同時依賴**舊狀態與舊輸入**——也就是讓輸出依賴**狀態之間的轉移**，而不只是狀態本身。



箭頭上的「1/0」表示：**輸入 1 時發生此轉移，輸出 0**。這稱為 **Mealy 機（Mealy machine）**。Moore 機與 Mealy 機都是**有限狀態機（finite state machine, FSM）**的例子。

實作這台 Mealy 機所需的真值表：

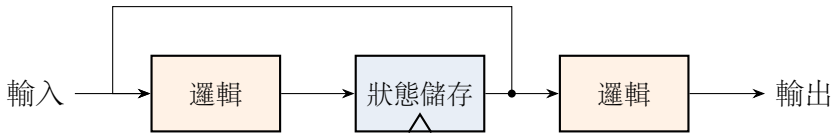
S_{old}	in_{old}	S_{new}	out
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	0

$$S_{\text{new}} = in_{\text{old}}, \qquad out = \neg S_{\text{old}} \wedge in_{\text{old}}.$$

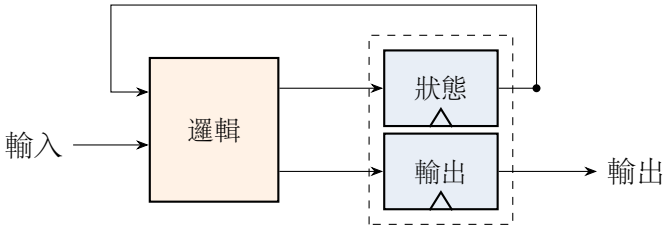
狀態的意義簡化成「**上一拍的輸入**」；輸出 =「上一拍輸入是 1、而再上一拍不是」——正是**上升緣偵測**！成本：**1 個正反器 + 1 個 AND + 1 個 NOT**（Moore 版要 2 個正反器與更多邏輯）。好多了！還能再省：用狀態正反器的 Q' 輸出取代 NOT 閘。

1.7 通用的 Moore ／ Mealy 架構

任何 Moore ／ Mealy 機都能用下列架構實作：



每一台 Moore 機



儲存（皆有時脈）

每一台 Mealy 機

FSM 實作守則

儲存可以用正反器或暫存器，但必須**有時脈（clocked）**。**邏輯**必須是組合邏輯（無時脈的閘），而且你可以像單觸發那樣，**把真值表寫下來**直接實作。（Mealy 機把輸出也存進有時脈的正反器，輸出因此能依賴「上一拍的輸入」——即狀態轉移。）

延伸補充：Moore vs. Mealy 的取捨與狀態編碼。

- Mealy 機通常**狀態較少**、硬體較省，且輸出對輸入的反應快一拍；Moore 機的輸出只隨狀態變化，更**同步、可預測**，較不易受輸入毛刺（glitch）影響。口訣：*Moore has more states*。
- **狀態編碼**：N 個狀態用 $\lceil \log_2 N \rceil$ 個位元是**二進位編碼**（暫存器最省、解碼邏輯較複雜）；**one-hot 編碼**則用 N 個正反器、每個狀態恰一個位元為 1（正反器較多、解碼邏輯極簡，FPGA 上常用——因為 FPGA 正反器多得是）。

1.8 不那麼正式的狀態圖

假設要做一台**販賣機**：(a) 等使用者輸入零食編號的第一位數字；(b) 等第二位數字，或按「返回」清除第一位；(c) 等投入足夠的錢，或按「返回」取消交易；(d) 出貨（若沒按「返回」）並找零，回到 (a)。

這不會是「純」Moore 或 Mealy 機——例如金額要用**暫存器**追蹤，而不是把每種金額都畫成一個狀態。但我們仍然可以、也應該用**追蹤內部狀態**的方式來蓋這個電路。

有限狀態機在**寫程式**時也非常有用，可以大幅簡化複雜的控制邏輯——例如平台遊戲 *Celeste* 的角色移動邏輯（原始碼已公開）就是圍繞一台有限狀態機組織的！

本章重點

- 抽象成元件：plexers、加法器、暫存器、閘——別再數 NAND 了。
- 計數器 = 暫存器 + +1 加法器 + reset MUX；「存狀態 → 組合邏輯算下一狀態 → 餵回」是萬用模式。
- 單觸發：長輸入變一拍脈衝；用狀態圖設計（等 1 ／送脈衝／等 0）。
- Moore 機：輸出 = f(狀態)；Mealy 機：輸出 = f(狀態, 輸入)，常可省狀態（單觸發從 2 個正反器降到 1 個）。
- 通用架構：時脈儲存 + 組合邏輯 + 回饋；照真值表實作。
- 販賣機等實務系統用「狀態 + 暫存器」的混合設計；FSM 在軟體中同樣好用。

2　電晶體（Transistors）

2.1　從類比到數位

到目前為止我們把電子訊號當成離散的 ON ／ OFF。但現實中它們是**連續的物理量**（例如導線上的電壓）：邏輯 0 用 0V 表示、邏輯 1 常用 5V 表示……那電壓是 0.0001V 或 4.9999V 怎麼辦?

由於電訊號本質連續、必有微小變異，實際上我們把**一段（小）電壓範圍**視為邏輯 0 或邏輯 1。不過我們對精確物理不感興趣，所以把這種連續邏輯抽象掉，改用離散變數思考:

定義

V_{dd}（drain 端電壓）代表可接受為**邏輯 1** 的電壓範圍；V_{ss}（source ／接地端電壓）代表可接受為**邏輯 0** 的電壓範圍。

隨著電晶體愈做愈小，V_{dd} 也逐步調低——省電、並避免過載燒壞它們!（5V 是早期標準；現代 CPU 核心電壓已降到 1V 上下。）

2.2　矽與半導體

定義

電晶體（transistor）是**電控開關**——用電訊號控制「通／斷」的開關。它們便宜、微小、可靠，因此成為現代電腦的基本積木。

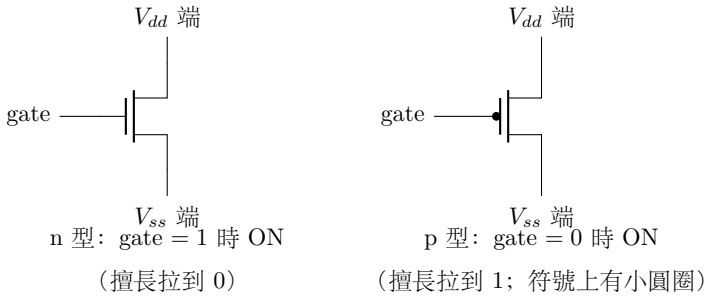
兩大類電晶體：**BJT**（雙極性接面電晶體）與 **MOSFET**（金氧半場效電晶體）——本課程看後者。

MOS 電晶體由**矽（silicon）**製成。矽是**半導體（semiconductor）**，也是地球上含量第二豐富的元素。半導體有個獨特性質：**某些條件下導電、其他條件下絕緣**（阻斷電流）。透過**摻雜（doping）**引入雜質，可以改變「需要什麼條件才會形成導電通道」。

2.3　n 型與 p 型電晶體

摻雜可以給矽結構**多餘的電子**或**電洞（hole）**（「該有電子的位置缺了一個電子」），造出兩種 MOS 電晶體:

n 型（nMOS）	**p 型（pMOS）**
摻雜成有**多餘電子**。稱為 negative 型——電子帶負電。	摻雜成有**電洞**。稱為 positive 型——電洞代表「缺少負電荷」。
閘極（gate）收到 V_{dd} ／ 1 時形成導電通道（**ON**）；gate = 0 時 OFF。	閘極收到 V_{ss} ／ 0 時形成導電通道（**ON**）；gate = 1 時 OFF。
擅長傳遞 V_{ss} ／ 0 訊號，傳 1 很差。	擅長傳遞 V_{dd} ／ 1 訊號，傳 0 很差。



兩型電晶體的互補分工

型別	gate = 0	gate = 1	擅長傳遞
n 型（nMOS）	OFF	ON	V_{ss} ／ 0
p 型（pMOS）	ON	OFF	V_{dd} ／ 1

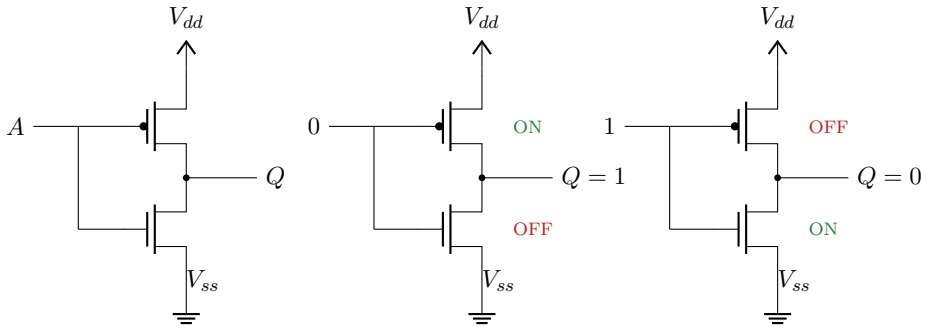
兩者對 gate 訊號的反應相反，擅長傳的訊號也相反——天生一對。

2.4　CMOS 邏輯

定義

CMOS（Complementary Metal-Oxide-Semiconductor，互補式金氧半）是多數現代電腦使用的技術：n 型與 p 型電晶體**協同工作**——**n 型**負責在 ON 時把輸出**接到** V_{ss}（下拉網路），**p 型**負責在 ON 時把輸出**接到** V_{dd}（上拉網路）。這種互補動作讓電路能高效切換，這正是「C」的由來。

最簡單的例子：**CMOS NOT 閘**（反相器）——一個 pMOS 疊在一個 nMOS 上，兩個 gate 接在一起當輸入：



- $A = 0$：pMOS ON（接通 V_{dd}）、nMOS OFF $\Rightarrow Q = 1$；
- $A = 1$：pMOS OFF、nMOS ON（接通 V_{ss}）$\Rightarrow Q = 0$。

注意：**必須有到 V_{ss} 的連接**——沒有連接 = 沒有訊號！（輸出任何時刻都必須被上拉網路或下拉網路**恰好其中之一**驅動。）

本章重點

- 電壓連續，用範圍抽象成離散：V_{dd} = 邏輯 1、V_{ss} = 邏輯 0；電晶體變小 ⇒ V_{dd} 降低（省電、防過載）。
- 電晶體 = 電控開關；矽半導體 + 摻雜（多電子 → n 型；電洞 → p 型）。
- nMOS：gate = 1 ON、擅長傳 0；pMOS：gate = 0 ON、擅長傳 1。
- CMOS：p 型上拉到 V_{dd}、n 型下拉到 V_{ss}，互補分工；NOT 閘 = 1 pMOS + 1 nMOS。

3　NAND vs NOR：用電晶體蓋邏輯閘

3.1　CMOS 閘的設計配方

要用電晶體蓋更複雜的邏輯，我們要設計**兩條路徑**：

- **上拉網路（pull-up network）**：用 **p 型**電晶體，定義「什麼時候輸出接到 V_{dd}（輸出 1）」；
- **下拉網路（pull-down network）**：用 **n 型**電晶體，定義「什麼時候輸出接到 V_{ss}（輸出 0）」。

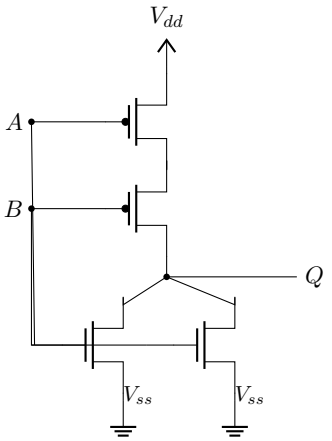
兩條路徑必須**互補**：任何輸入組合下，恰好一條形成完整導電通道——輸出永遠是明確的 0 或 1。

工具箱：電晶體**串聯** = 「兩個都 ON 才通」（AND 條件）；**並聯** = 「任一個 ON 就通」（OR 條件）。

3.2　NOR 閘

真值表：只有 $A = B = 0$ 時輸出 1。所以上拉（輸出 1）的條件是「$A = 0$ **且** $B = 0$」⇒ 兩個 pMOS **串聯**；下拉（輸出 0）的條件是「$A = 1$ **或** $B = 1$」⇒ 兩個 nMOS **並聯**：

A	B	$A \downarrow B$
0	0	1
0	1	0
1	0	0
1	1	0



逐一檢查四種輸入（ON/OFF 狀態）：

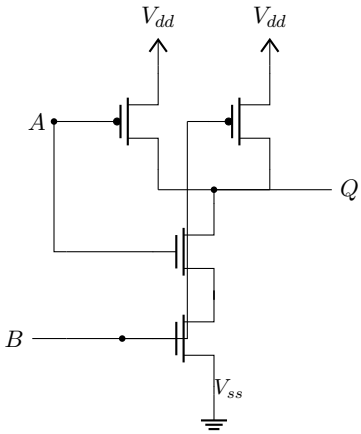
(A, B)	pMOS_A	pMOS_B	nMOS_A	nMOS_B	Q
$(0, 0)$	ON	ON	OFF	OFF	上拉通 ⇒ **1**
$(0, 1)$	ON	OFF	OFF	ON	下拉通 ⇒ **0**
$(1, 0)$	OFF	ON	ON	OFF	下拉通 ⇒ **0**
$(1, 1)$	OFF	OFF	ON	ON	下拉通 ⇒ **0**

任何時刻**恰好一條**路徑導通——互補成立 ✓。

3.3　NAND 閘

同樣的流程：只有 $A = B = 1$ 時輸出 0。上拉條件「$A = 0$ **或** $B = 0$」⇒ 兩個 pMOS **並聯**；下拉條件「$A = 1$ **且** $B = 1$」⇒ 兩個 nMOS **串聯**：

A	B	$A \uparrow B$
0	0	1
0	1	1
1	0	1
1	1	0



上拉：pMOS **並聯**
（A=0 或 B=0 就輸出 1）

下拉：nMOS **串聯**
（A=1 且 B=1 才輸出 0）

(A, B)	pMOS_A	pMOS_B	nMOS_A	nMOS_B	Q
$(0,0)$	ON	ON	OFF	OFF	上拉通 $\Rightarrow$ **1**
$(0,1)$	ON	OFF	OFF	ON	上拉通 $\Rightarrow$ **1**
$(1,0)$	OFF	ON	ON	OFF	上拉通 $\Rightarrow$ **1**
$(1,1)$	OFF	OFF	ON	ON	下拉通 $\Rightarrow$ **0**

CMOS 閘設計對照

閘	上拉（pMOS）	下拉（nMOS）	電晶體數
NOT	單一	單一	2
NAND	**並聯**	**串聯**	4
NOR	**串聯**	**並聯**	4
AND（= NAND+NOT）	—	—	6
OR（= NOR+NOT）	—	—	6

規律：上拉與下拉網路永遠是**對偶**的（串聯 ↔ 並聯互換）。CMOS 天生擅長**反相**函數（NAND ／ NOR ／ NOT）——AND、OR 反而要多花一個反相器。

3.4　結構速度：為什麼 NAND 是工業標準?

NAND 和 NOR 都用「2 顆並聯 + 2 顆串聯」的相似設計，為什麼業界標準是 NAND? 兩個物理事實：

1. **串聯比並聯慢**：電流要**依序穿過兩顆**電晶體才能到達輸出（電阻加倍）；並聯只要穿過一顆。
2. **p 型比 n 型慢**：電洞在矽晶格中的移動速度 **比電子慢**（載子遷移率低，約差 2–3 倍）。

把兩張表疊起來看：

	NAND	NOR
pMOS（慢）	並聯（快的接法）✓	**串聯（慢的接法）✗**
nMOS（快）	串聯（慢的接法，但元件快）	並聯（快的接法）

NOR 的問題：V_{dd} 訊號要穿過**串聯的 p 型**電晶體——**最慢的元件用最慢的接法**，是全場最糟的組合。NAND 則把串聯留給快的 n 型、讓慢的 p 型並聯——兩害相權取其輕。

定義

關鍵路徑（critical path）：輸入到輸出之間**延遲最長**的那條路徑，它決定了整個電路的切換速度。NOR 的關鍵路徑（串聯 pMOS 上拉）比 NAND 的長，所以 **NAND 是業界偏好的閘**。

延伸補充：邏輯努力（logical effort）量化這件事。以標準尺寸比較，2 輸入 NAND 的邏輯努力 $g = 4/3$、NOR 為 $g = 5/3$（越小越快）；n 輸入時 NAND 為 $(n+2)/3$、NOR 為 $(2n+1)/3$——差距隨輸入數擴大。還有製造上的好處：NAND 各電晶體尺寸可以做得接近一致，NOR 則需要把串聯的 pMOS 加大好幾倍來補償，佔面積又增加輸入電容。這也呼應第一週的結論：NAND 功能完備——「一種閘打天下」時，選最快最省的那種。

本章重點

- CMOS 閘 = pMOS 上拉網路 + nMOS 下拉網路，兩者邏輯互補、結構對偶。
- NOR：pMOS 串聯、nMOS 並聯；NAND：pMOS 並聯、nMOS 串聯；各 4 顆。
- 串聯慢於並聯；p 型（電洞）慢於 n 型（電子）。
- NOR 把「慢元件」用「慢接法」（串聯 pMOS）⇒ 關鍵路徑最長；NAND 避開了這個組合 ⇒ 工業標準。

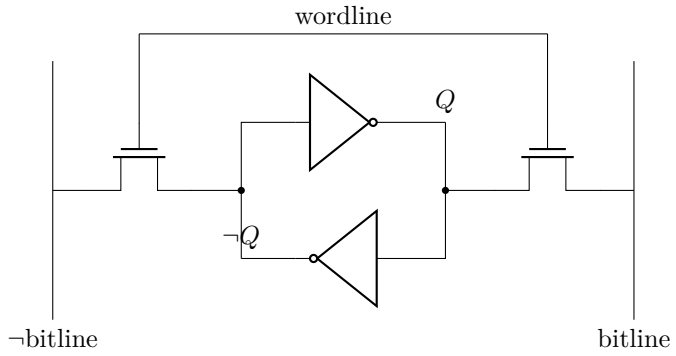
4　SRAM vs DRAM：記憶胞的內部構造

4.1　SRAM 記憶胞

定義

SRAM（Static Random Access Memory，靜態隨機存取記憶體）：只要持續供電，值就一直保持——所以叫**靜態**。

SRAM 記憶胞的核心是**兩個交叉耦合的反相器（cross-coupled inverters）**：Q 餵進一個反相器產生 $\neg Q$、$\neg Q$ 再餵回另一個反相器產生 Q……形成**雙穩態（bistable）**迴圈，位元值就「卡」在裡面（若 $Q = 0$ 則 $\neg Q = 1$、於是 $Q = 0$……自我維持）。這跟第三週 NAND 閂鎖的回饋原理一模一樣！



一個 SRAM 記憶胞有兩條輸出線：**bitline** 與 **¬bitline**。當 **wordline** 輸入為高，它連接的兩顆 n 型**存取電晶體**都導通，資料值就能**傳入或傳出** bitlines（讀與寫共用同一組線）。

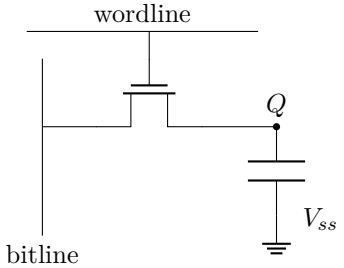
完整電路裡每個反相器就是第 2 章的 CMOS NOT（2 顆電晶體），所以標準 SRAM 記憶胞共 $2 \times 2 + 2 = 6$ 顆電晶體——即業界所稱的 **6T cell**。

4.2　DRAM 記憶胞

定義

DRAM（Dynamic Random Access Memory，動態隨機存取記憶體）：值只能保存**一小段時間**，之後必須**更新（refresh）**——所以叫**動態**。

位元值存在一顆**電容（capacitor）**上：充電 = 存 1、放電 = 存 0。一顆 n 型電晶體把電容接上 bitline：



只有 **1 顆電晶體 + 1 顆電容**（**1T1C**）——比 SRAM 的 6 顆便宜、緊湊得多。但代價是：

- 導通電晶體去**讀或寫**時，電容會**放掉一些電**（讀取是破壞性的，讀完要寫回）；
- 就算不碰它，電晶體關著也會**漏電流**——所以即使不存取，也必須**定期 refresh**（每隔幾毫秒充回去）。

4.3　三種揮發性記憶體的比較

我們已看過三種揮發性儲存：**正反器、SRAM、DRAM**。怎麼比較？

	正反器	SRAM	DRAM
每位元成本	12–30 顆電晶體	4–6 顆電晶體	1 顆電晶體 + 1 電容
存取速度	最快（輸出即資料）	介於中間	最慢
需要 refresh	否	否	是
相對單價／面積	最貴、最大	中	最便宜、最省
典型用途	CPU 內部暫存器	快取（cache）	主記憶體

- **正反器**：資料位元**立即**出現在輸出——最快；但要 12–30 顆電晶體，成本、功耗、面積都貴。
- **DRAM**：延遲比 SRAM 長，因為 bitline **沒有電晶體主動驅動**——必須等電荷（相對緩慢地）從電容流到 bitline；但 1T1C 便宜又緊湊。
- **SRAM**：4–6 顆電晶體，存取時間介於正反器與 DRAM 之間。

哪種記憶體「最好」？——**取決於設計的速度、成本與功耗需求**！（這正是下週記憶體階層的伏筆：三種全都用，各司其職。）

延伸補充：把第三週與本週串起來。第三週說「SRAM 只需 4–6 顆電晶體、我們的正反器做法要 ~40 顆」——現在你知道原因了：SRAM 直接用**兩個交叉耦合反相器**存位元（6T），而不是蓋出完整的邊緣觸發 D 正反器。DRAM 的「每隔幾 ms 更新 + 每次讀取後重寫」也有了物理解釋：**電容漏電**與**破壞性讀取**。記憶體階層（暫存器 → SRAM 快取 → DRAM 主記憶體）的本質，就是用「速度 vs. 密度／成本」的物理取捨換取整體效能。

本章重點

- SRAM 記憶胞：交叉耦合反相器（雙穩態）+ 2 顆存取電晶體 = 6T；wordline 開門、bitline ／ ¬bitline 進出資料；不需 refresh。
- DRAM 記憶胞：1T1C，電容電荷存位元；讀取破壞性、電晶體漏電 ⇒ 必須定期 refresh。
- 速度：正反器 > SRAM > DRAM；成本／密度恰好相反。
- 選擇取決於速度、成本、功耗需求——沒有萬能的記憶體。

5　綜合練習題（附詳解）

練習 1：計數器追蹤

一個 4 位元計數器（行為同第 1 章）初始值為 `1101`，連續五個上升緣時 *reset* 依序為 $0, 0, 1, 0, 0$。寫出每個上升緣之後的輸出（二進位與十進位）。

解答

- 緣 1（*reset*=0）：$1101 + 1 = 1110$（14）；
- 緣 2（*reset*=0）：1111（15）；
- 緣 3（*reset*=1）：歸零 $\rightarrow$ 0000（0）；
- 緣 4（*reset*=0）：0001（1）；
- 緣 5（*reset*=0）：0010（2）。

補充：若第 3 緣沒有 reset，$1111 + 1 = 1\,0000$ 丟棄進位後會**繞回** 0000——計數器本身就是模 2^N 算術。

練習 2：幫計數器加功能

替計數器加一個輸入 *load* 與資料輸入 *in*（優先序：*reset* > *load* > 計數）。寫出非正式真值表，並說明硬體上要怎麼改。

解答

reset	*load*	out_{new}
1	X	`0x0000`
0	1	*in*
0	0	$out_{\text{old}} + 1$

硬體：**再串一顆 MUX**。第一顆 MUX 用 *load* 在「$out_{\text{old}} + 1$」與「*in*」之間選；第二顆用 *reset* 在第一顆的輸出與常數 0 之間選（reset 優先，放最後）。這正是 Hack CPU 程式計數器（PC）的雛形：inc ／ load ／ reset 三種模式層層疊 MUX。

練習 3：單觸發 Moore 機追蹤

Moore 版單觸發（$S0$=00、$S1$=01、$S2$=10，公式見第 1 章）初始在 $S0$，輸入序列 $in = 1, 1, 1, 0, 1, 0$（每拍一個）。寫出每一拍結束後的狀態與該拍輸出。

解答

用轉移規則逐拍推（輸出看**當拍所在**的狀態）：

拍	1	2	3	4	5	6
拍首狀態	$S0$	$S1$	$S2$	$S2$	$S0$	$S1$
in	1	1	1	0	1	0
輸出（狀態函數）	0	**1**	0	0	0	**1**
拍末狀態	$S1$	$S2$	$S2$	$S0$	$S1$	$S0$

兩段連續的 1（長度 3 與長度 1）各只產生一**拍**輸出 1 ✓ ——單觸發行為正確。

練習 4：驗證單觸發的邏輯式

用第 1 章的轉移真值表逐列驗證 $X_{\text{new}} = (X_{\text{old}} \vee Y_{\text{old}}) \wedge in$ 與 $Y_{\text{new}} = \neg X_{\text{old}} \wedge \neg Y_{\text{old}} \wedge in$。

解答

X_{o}	Y_{o}	in	$(X_{\text{o}} \vee Y_{\text{o}}) \wedge in$	表中 X_{new}	$\neg X_{\text{o}} \wedge \neg Y_{\text{o}} \wedge in$	表中 Y_{new}
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	1	1	1	0	0

六個有效列全部吻合 ✓（$X_{\text{o}}Y_{\text{o}} = 11$ 是 don't care，不必檢查）。直覺：$Y_{\text{new}} = 1$ 唯讀「$S0$ 看到 1」；$X_{\text{new}} = 1$ 唯讀「$S1$ 或 $S2$ 看到 1」。

練習 5：設計一台 Moore 機——連續兩個 1 偵測器

設計 Moore 機：輸入序列中**最近兩拍都是 1** 時輸出 1。畫狀態圖、選編碼、寫轉移／輸出真值表並化簡。

解答

三個狀態：$S0$ =「上一拍是 0」（輸出 0）、$S1$ =「上一拍是 1 但再上一拍不是」（輸出 0）、$S2$ =「最近兩拍都是 1」（輸出 1）。轉移：任何狀態看到 0 回 $S0$；$S0 \xrightarrow{1} S1$；$S1 \xrightarrow{1} S2$；$S2 \xrightarrow{1} S2$。編碼 $S0$=00、$S1$=01、$S2$=10（XY）：

X_{o}	Y_{o}	in	X_{n}	Y_{n}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	X	X	X

化簡（同單觸發的形狀!）：$X_{\mathrm{n}} = (X_{\mathrm{o}} \vee Y_{\mathrm{o}}) \wedge in$、$Y_{\mathrm{n}} = \neg X_{\mathrm{o}} \wedge \neg Y_{\mathrm{o}} \wedge in$、$out = X$。有趣的觀察：轉移邏輯與單觸發**完全相同**，只差輸出函數（單觸發取 Y、本題取 X）——同一台機器骨架能長出不同行為。

練習 6：Mealy 版連續兩個 1 偵測器

用 Mealy 機重做練習 5，比較狀態數。

解答

狀態只需記「上一拍輸入」：$S \in \{0,1\}$（1 個正反器）。輸出 =「上一拍是 1 **且**這一拍也是 1」：

$$S_{\mathrm{new}} = in, \qquad out = S_{\mathrm{old}} \wedge in.$$

狀態圖：$S0 \xrightarrow{1/0} S1$、$S1 \xrightarrow{1/1} S1$、$S1 \xrightarrow{0/0} S0$、$S0 \xrightarrow{0/0} S0$。
比較：Moore 版 3 個狀態（2 個正反器）；Mealy 版 2 個狀態（1 個正反器）——再次印證「Mealy 常可省狀態」。代價：Mealy 的輸出在**同一拍**直接受 in 影響（組合路徑），時序分析要更小心。

練習 7：Moore vs. Mealy 觀念題

(a) 兩者輸出各依賴什麼？(b) 為什麼說 Mealy 的輸出「快一拍」？(c) 什麼情況偏好 Moore？

解答

(a) Moore：輸出 $= f$(狀態)；Mealy：輸出 $= f$(狀態, 輸入)。
(b) Moore 機要「先轉移到新狀態、下一拍輸出才反映」；Mealy 機的輸出直接含輸入項，**輸入一變、同拍的輸出就能變**——效果上早一個時脈週期。
(c) 需要輸出**嚴格同步、無毛刺**時（輸出只在時脈緣後改變、期間穩定），或者狀態語意要清楚（每個狀態自帶輸出、好除錯）。課程單觸發的 Mealy 版把輸出也存進正反器，正是兼取兩者優點的做法。

練習 8：電晶體開關行為

(a) 完成表格：nMOS ／ pMOS 在 gate = 0/1 時的 ON/OFF。(b) 為什麼 CMOS 用 pMOS 上拉、nMOS 下拉，而不是反過來?

解答

(a) nMOS：gate = 0 OFF、gate = 1 ON；pMOS：gate = 0 ON、gate = 1 OFF。

(b) 因為**各自擅長傳的訊號不同**：p 型擅長傳 V_{dd} ／ 1、n 型擅長傳 V_{ss} ／ 0，各自傳相反訊號時會劣化（電壓降）。讓 pMOS 專職「把輸出拉到 1」、nMOS 專職「把輸出拉到 0」，每顆電晶體都只做自己擅長的事——這就是「互補（Complementary）」的意義。

練習 9：設計 3 輸入 NAND 的電晶體電路

描述 3 輸入 NAND 的上拉與下拉網路，並給出電晶體數。一般的 n 輸入 NAND 呢?

解答

輸出為 0 僅當 $A = B = C = 1 \Rightarrow$ 下拉：**3 顆 nMOS 串聯**；輸出為 1 當任一輸入為 0 $\Rightarrow$ 上拉：**3 顆 pMOS 並聯**。共 **6 顆**。

一般 n 輸入 NAND（或 NOR）：n 顆 pMOS + n 顆 nMOS = $2n$ 顆。注意串聯堆疊越高越慢——實務上超過 3–4 輸入的閘常拆成多級小閘。

練習 10：AND 閘為什麼要 6 顆電晶體?

(a) 為什麼 CMOS 不能用 4 顆電晶體直接做 AND? (b) 那 AND 怎麼做? (c) 由此解釋第一週「NAND 是天然積木」的深層原因。

解答

(a) CMOS 結構天生**反相**：pMOS 上拉網路在輸入「低」時導通、nMOS 下拉在輸入「高」時導通，所以單級 CMOS 閘的輸出必然是輸入的**反函數**（輸入全高 $\Rightarrow$ 輸出低）。AND 是非反相函數，單級做不出來。

(b) AND = NAND + NOT = 4 + 2 = 6 顆。（OR 同理 = NOR + NOT。）

(c) 在 CMOS 世界裡，**反相閘（NAND ／ NOR ／ NOT）才是原生元件**，AND ／ OR 反而是二手貨。既然一定要選一種反相閘當標準積木，就選最快的——NAND（見第 3 章）。第一週的「NAND 功能完備」加上本週的「NAND 物理上最快」，兩條線在此會合。

練習 11：NAND vs NOR 速度論證

不看講義，完整重建「NAND 比 NOR 快」的論證鏈。

解答

1. 物理事實一：**電洞遷移率低於電子** ⇒ pMOS 天生比 nMOS 慢。
2. 物理事實二：**串聯要穿過兩顆電晶體**（電阻相加）⇒ 串聯比並聯慢。
3. NOR 結構：上拉 = **pMOS 串聯**——慢元件 × 慢接法，輸出拉到 1 特別慢。
4. NAND 結構：串聯的是**快的 nMOS**、慢的 pMOS 走並聯——避開最糟組合。
5. 結論：NOR 的**關鍵路徑**（最長延遲路徑）比 NAND 長，NAND 切換更快 ⇒ 工業標準。（量化版：2 輸入 NAND 邏輯努力 $4/3 <$ NOR 的 $5/3$。）

練習 12：SRAM ／ DRAM 觀念題

(a) SRAM 記憶胞如何存住一個位元？wordline 的角色？(b) DRAM 為什麼必須 refresh？兩個原因。(c) 為什麼 DRAM 讀取比 SRAM 慢？(d) 一顆 16 Ki × 8 位元的 SRAM 晶片（6T cell），存儲陣列約需多少電晶體？換成 DRAM 呢？

解答

(a) 兩個**交叉耦合反相器**形成雙穩態迴圈（Q 與 $\neg Q$ 互鎖），位元自我維持。wordline 拉高時，兩顆 n 型存取電晶體導通，讓 bitline ／ ¬bitline 能**讀出或寫入**這個迴圈。
(b) 一、**漏電流**：電晶體即使關閉也漏電，電容電荷會流失（每隔幾 ms 要充回）；二、**破壞性讀取**：讀取時電容對 bitline 放電，讀完必須重寫。
(c) DRAM 的 bitline **沒有電晶體主動驅動**，要等電荷從小電容緩慢流到 bitline（再由感測放大器判讀）；SRAM 的反相器則主動驅動 bitline。
(d) SRAM：$16384 \times 8 \times 6 = 786{,}432$ 顆電晶體。DRAM：$16384 \times 8 \times 1 = 131{,}072$ 顆電晶體 + 同數量的電容——約 6 倍的密度差距，這就是主記憶體用 DRAM 的原因。

A 附錄：速查表

A.1 FSM 設計流程

1. 畫**狀態轉移圖**（規格的抽象實作）；
2. 選**狀態編碼**（二進位／ one-hot ／ Gray⋯）；
3. 寫**轉移與輸出真值表**（不存在的狀態填 X）；
4. **化簡**邏輯（K-map 或觀察）；
5. 接線：時脈儲存 + 組合邏輯 + 回饋。

A.2 CMOS 閘電晶體數

閘	NOT	NAND	NOR	AND	OR
電晶體數	2	4	4	6	6
結構	1P+1N	並 P+ 串 N	串 P+ 並 N	NAND+NOT	NOR+NOT

n 輸入 NAND ／ NOR：$2n$ 顆。CMOS 單級只能做**反相**函數。

A.3 記憶體技術對照

	正反器	SRAM（6T）	DRAM（1T1C）
儲存機制	閘迴圈	交叉耦合反相器	電容電荷
速度	最快	中	最慢
refresh	不用	不用	每幾 ms + 讀後重寫
密度	最低	中	最高
用途	暫存器	快取	主記憶體

A.4 名詞中英對照

英文	中文	英文	中文
component / subcircuit	元件／子電路	doping	摻雜
counter	計數器	hole	電洞
one-shot	單觸發	nMOS / pMOS	n 型／ p 型電晶體
state diagram	狀態圖	CMOS	互補式金氧半
finite state machine	有限狀態機	pull-up / pull-down	上拉／下拉網路
Moore / Mealy machine	Moore ／ Mealy 機	series / parallel	串聯／並聯
state encoding	狀態編碼	critical path	關鍵路徑
one-hot encoding	one-hot 編碼	logical effort	邏輯努力
transistor	電晶體	wordline / bitline	字線／位元線
semiconductor	半導體	access transistor	存取電晶體
V_{dd} / V_{ss}	高／低電位軌	capacitor	電容
MOSFET / BJT	金氧半場效／雙極性	refresh	更新
carrier mobility	載子遷移率	destructive read	破壞性讀取

A.5 參考資料

- John Lapinskas, *Building with flip-flops and registers*（4-1），Kira Clements, *Transistors ／ NAND vs NOR ／ SRAM vs DRAM*（4-2 至 4-4），COMSM1302, University of Bristol.
- Nisan & Schocken, *The Elements of Computing Systems*（nand2tetris），Ch. 3（Counter ／ PC 晶片）.
- MIT 6.111 *Introductory Digital Systems Laboratory*: Finite State Machines 講義（狀態編碼、Moore/Mealy level-to-pulse）.
- Harris & Harris, *Digital Design and Computer Architecture*——CMOS 閘設計與 FSM 章節.
- Wikipedia：*Logical effort*、*CMOS*、*Static random-access memory*、*Dynamic random-access memory*.
- Maddy Thorson & Noel Berry, *Celeste* 玩家控制原始碼（公開的 Player.cs，FSM 實例）.