計算機組織與 RISC-V CPU 設計 · 第 7 章

輸入/輸出

I/O 模組 / 程式化 I/O / 中斷驅動 I/O / DMA / I/O 通道與外部介面

摘要

I/O 是計算機三大子系統中最「雜」的一塊: 週邊裝置種類繁多、速度差異可達九個數量級、資料格式各異。本章說明為何需要 **I/O** 模組作為週邊與系統之間的中介, 然後依「處理器介入程度遞減」的順序, 詳述三種 I/O 技術: 程式化 **I/O**、中斷驅動 **I/O** 與直接記憶體存取 **(DMA)**, 最後介紹 I/O 通道/處理器的概念與外部介面的演進。本章對應 Stallings 第 7 章。

目錄

1 外部裝置與 I/O 模組

1.1 為何不把週邊直接接上匯流排

1. 週邊種類繁多、操作方法各異, 處理器無法內建所有裝置的控制邏輯;
2. 週邊速度通常遠慢於處理器與記憶體, 直接掛上高速匯流排並不實際;
3. 週邊的資料格式與字長常與主機不同。

因此需要 **I/O** 模組: 對內以標準方式連接系統匯流排, 對外以裝置專屬的訊號連接一個或多個週邊。I/O 模組的主要功能:

- 控制與時序: 協調內部資源與外部裝置間的資料流;
- 處理器通訊: 命令解碼、資料交換、狀態回報 (READY/BUSY/錯誤)、位址識別;
- 裝置通訊: 命令、狀態與資料;
- 資料緩衝: 調和速度差 (記憶體以高速率突發、裝置以低速率涓流);
- 錯誤偵測: 機械故障、傳輸位元錯誤 (同位檢查) 等。

1.2 I/O 定址: 記憶體映射 vs. 隔離式

- 記憶體映射 **I/O(memory-mapped)**: 裝置暫存器佔用記憶體位址空間的一段, 用一般載入/儲存指令存取。**RISC-V** 採此方式——沒有專用 I/O 指令,`lw`/`sw` 到裝置位址即可 (第 21 章的 UART 範例)。
- 隔離式 **I/O(isolated)**: 獨立的 I/O 位址空間與專用指令 (x86 的 `IN`/`OUT`)。

2 程式化 I/O(Programmed I/O)

處理器執行 I/O 指令後, 以迴圈輪詢 **(polling)** 狀態暫存器直到裝置完成:

```
# s0 = UART 基底位址; 偏移 0=資料, 4=狀態(bit0=TX Ready)
poll:
    lw   t0, 4(s0)        # 讀狀態暫存器
    andi t0, t0, 1        # 檢查 TX Ready 位元
    beqz t0, poll         # 未就緒 -> 繼續輪詢 (忙碌等待!)
    sb   a0, 0(s0)        # 就緒 -> 寫資料暫存器
    ret
```

程式 1 程式化 I/O: 輪詢 UART 送出一個位元組

優點是簡單; 致命缺點是忙碌等待: 處理器速度被綁在最慢的裝置上, 絕大多數週期都在空轉。

3 中斷驅動 I/O

處理器發出 I/O 命令後繼續執行其他工作; 裝置就緒時發出中斷請求, 處理器在指令邊界回應: 保存脈絡 → 跳到中斷服務常式 (ISR)→ 傳輸一筆資料 → 恢復脈絡。

3.1 裝置識別

多裝置共用中斷時, 如何知道是誰?

- 多中斷線: 每裝置一條線; 線的數量有限。
- 軟體輪詢:ISR 逐一查詢各模組狀態; 慢。
- 菊鏈 **(**硬體輪詢/向量式**)**: 中斷確認訊號沿鏈傳遞, 請求者把向量 (自身識別) 放上匯流排; 處理器以向量索引跳到專屬 ISR。
- 匯流排仲裁: 模組先取得匯流排擁有權才可發中斷, 確認後放上向量。

現代實作 (如 RISC-V 的 PLIC 平台級中斷控制器) 綜合了優先權與向量的概念: 多個來源集中到控制器, 依優先權裁決後向 hart 發出單一中斷,ISR 讀取 claim 暫存器得知來源。

注意

中斷驅動 I/O 仍有極限: 每一筆資料都要經過處理器 (一次中斷、一次 ISR)。對高速裝置 (磁碟、網路) 而言, 中斷率會淹沒處理器。

4 直接記憶體存取 (DMA)

4.1 運作方式

DMA 模組是一個能自行主導匯流排的專用控制器。處理器要傳一個區塊時, 只需告訴 DMA: 讀或寫、裝置位址、記憶體起始位址、字數; 然後整個區塊的搬移由 **DMA** 完成**,** 完全不經過處理器, 結束時 DMA 發一次中斷。

程式化 I/O: 每字都要處理器 → 中斷式: 每字一次中斷 → DMA: 每區塊一次中斷

4.2 週期竊取

DMA 與處理器共用匯流排:DMA 需要傳輸時「竊取」一個匯流排週期 (cycle stealing), 處理器若正巧要用匯流排則等待一個週期——這不是中斷, 不需保存脈絡, 代價遠小於中斷。處理器只在匯流排存取上稍微變慢。

DMA 的配置可以是: 單一匯流排上的獨立 DMA 模組;DMA 與 I/O 模組整合; 或 DMA 與 I/O 裝置間有專屬 I/O 匯流排。後兩者可把 DMA 與裝置間的傳輸移出系統匯流排, 進一步減少干擾。

5 I/O 通道與 I/O 處理器

演進的最後一步是讓 I/O 模組成長為可執行程式的處理器:

- **I/O** 通道 **(channel)**: 能執行記憶體中的「通道程式」, 自主完成多筆傳輸、位址轉換、錯誤處理; 大型主機的傳統。選擇器通道服務單一高速裝置; 多工器通道分時服務多個低速裝置 (位元組多工) 或高速裝置 (區塊多工)。
- **I/O** 處理器: 更進一步, 有自己的本地記憶體, 可控制大量裝置。現代網卡 (具 TCP offload)、NVMe 控制器、GPU 的 copy engine 都是這個概念的後裔。

6 外部介面

Stallings 第 6 版介紹的 FireWire(IEEE 1394) 與 InfiniBand 展示了外部介面從平行走向高速串列、封包化、交換式的趨勢。今日的對應者:

- **USB**(1.5 Mbps 到 USB4 的 80 Gbps): 取代了幾乎所有低中速週邊介面;
- **PCIe**: 內部高速串列互連,NVMe SSD、GPU 直連;
- **InfiniBand / RoCE**: 資料中心高效能互連, 支援 RDMA(遠端 DMA——DMA 概念的網路延伸);
- **SATA/SAS**: 儲存專用串列介面。

共同特徵: 差動訊號、點對點、通道聚合 (multi-lane)、封包協定分層——平行匯流排的時脈偏斜問題使「更多線」不再是提速的路。

7 本章重點回顧

本章要點

- I/O 模組是週邊與系統間的中介: 控制時序、通訊、緩衝、錯誤偵測。
- 三種技術依處理器介入遞減: 程式化 (忙碌等待)→ 中斷驅動 (每字一次中斷)→ DMA(每區塊一次中斷、週期竊取)。
- 中斷識別: 多線、軟體輪詢、菊鏈向量、匯流排仲裁; 現代用中斷控制器 (RISC-V:PLIC/CLINT)。
- I/O 通道/處理器讓 I/O 子系統自主執行程式;RDMA、智慧網卡是其現代形態。
- RISC-V 採記憶體映射 I/O: 以 lw/sw 存取裝置暫存器。

8 複習問題

1. 列舉三個「週邊不直接掛上系統匯流排」的理由。
2. 記憶體映射 I/O 與隔離式 I/O 的差異?RISC-V 採哪種?
3. 程式化 I/O 的效率問題出在哪? 中斷驅動 I/O 解決了什麼、又留下什麼問題?
4. DMA 的週期竊取與中斷有何本質不同? 為何 DMA 對高速裝置至關重要?
5. 磁碟以 10 MB/s 傳輸, 每筆 4 B: 純中斷式 I/O 每秒需處理多少次中斷? 若 ISR 開銷 2 us, 處理器多少比例時間耗在 I/O? 改用 DMA(64 KiB 區塊) 後呢?
6. 為什麼現代外部介面普遍從平行轉向高速串列?

參考資料

[1] W. Stallings, *Computer Organization and Architecture*, 6th ed., Chapter 7.

[2] RISC-V International, *RISC-V Platform-Level Interrupt Controller Specification*.

[3] USB Implementers Forum, https://www.usb.org