Lernzettel

Ein-/Ausgabeorganisation: Adressierung, Synchronisation, DMA, Direktzugriff, MMIO, Interrupts

Universität: Technische Universität Berlin
Kurs/Modul: Rechnerorganisation

Erstellungsdatum: September 20, 2025

Lernzettel: Ein-/Ausgabeorganisation: Adressierung, Synchronisation, DMA, Direktzugriff, MMIO, Interrupts

(1) Adressierung.
I/O-Adressierung erfolgt typischerweise durch zwei Konzepte:

- Speicheradressierbar (Memory-Mapped I/O, MMIO): Peripherie-Register befinden sich im normalen Speicheradressraum. Lese-/Schreibbefehle an Adressen dienen gleichzeitig dem Zugriff auf Speicher und auf die Peripherie.
- Port-Adressierung (Port IO, PIO): Peripherie hat einen separaten I/O-Raum; spezielle CPU-Instruktionen (z. B. IN/OUT) greifen dort zu. Typisch in klassischen Architekturen genutzt.

MMIO vs PIO:

- MMIO: einheitlicher Adressraum, einfache Adressierungslogik, oft einfachere Softwareabstraktion, größere Adressraume nötig; ggf. Cache-Kohärenzprobleme beachten.
- PIO: separater I/O-Raum, oft schnellere Zugriffe über spezialisierte In/Out-Befehle, geringerer Adressraumbedarf, aber spezialisierte Instruktionen erforderlich.

(2) Synchronisation.
Koordination zwischen CPU, Peripherie und ggf. anderen Prozessen:

- Polling (busy-wait) vs. Interrupt-getriebene Synchonisation: Polling ist einfach, aber CPU blockiert häufig.
- Handshake-Schemata: Status- und Acknowledge-Register zur Synchronisation zwischen Producer und Consumer.
- Pufferung/FIFO: kleine Zwischenpuffer ermöglichen Entkopplung von Produzenten und Konsumenten.
- Synchronisationsprimitive in Mehrprozessorsystemen: Semaphoren, Mutexe, Barrieren.

(3) DMA (Direct Memory Access).
Zweck: Entlastung der CPU bei Datentransfers zwischen Speicher und Peripherie.

- DMA-Controller übernimmt den Transfer; CPU wird frei für andere Aufgaben.
- Bus-Master-Funktionalität: DMA-Controller steuert den Bus, um Daten direkt zwischen Speicher und Peripherie zu verschieben.
- Transferarten: Burst-Modus (blocksweise), Cycle-Stealing (CPU-Benutzung wird in kleinen Intervallen gestattet).
- Bus-Arbitration: Wer darf den Bus nutzen, Konflikte vermeiden.
- Konsistenz/Warung: Cache-Kohärenz beachten; Write-Back/Write-Through-Strategien relevant.

(4) Direktzugriff (PIO).
Direkter Zugriff auf Peripherie-Register durch CPU über definierte Adressen.

- IO-Port-Zugriff: Nutzung von In/Out-Instruktionen (bei manchen Architekturen).
- Vorteile: einfache, schnelle Zugriffe ohne Umweg über DMA oder Treiber-Queues.
- Nachteile: CPU-bindend, geringe Kopieroptimierung, Skalierbarkeit begrenzt.

(5) MMIO (Memory-Mapped I/O).
Peripherie-Register sind im normalen Speicheradressraum adressierbar.

- Vorteile: einheitliche Adressierung, einfache Programmiermodelle, oft bessere Integration mit-cache-sicheren Zugriffsmechanismen.
- Nachteile: größerer Adressraumbedarf, potenzielle Cache-Probleme, nötige Schutzmechanismen (MMU/Cache-Coherence).

(6) Interrupts.
Unterbrechungen signalisieren asynchron Ereignisse von Peripherie an die CPU.

- Interrupt-Linien: hardwareseitige Signale, können level- oder edge-triggered sein.
- Interrupt-Handling: Interrupt-Controller, Interrupt-Vektor-Tabelle (z. B. IDT), Zuordnung von ISR-Adressen.
- Maskierung/Priorität: Maskierung von Interrupts, Prioritätsordnung, Nested Interrupts.
- Ablauf: Peripherie löst Interrupt aus → CPU speichert Kontext, springt zum ISR → ISR verarbeitet → Kontextwiederherstellung und Rückkehr zur normalen Ausführung.

Zusammenhänge zwischen Begriffen.
DMA kann Interaktionen mit MMIO-Register-Dateien ermöglichen, Interrupts können als Benachrichtigung nach fertigen DMA-Transfers dienen, sowie Synchronisation über FIFO-Puffer unterstützen. Direktzugriff (PIO) bietet einfache Wege, Status- bzw. Kontrollregister unmittelbar zu überprüfen oder zu setzen, während MMIO einen einheitlichen Zugriffspunkt bildet.