Kapitel 15
Input und Output

Input und Output

→ Verbindung zur Außenwelt !

Aufgaben sind z.B.:

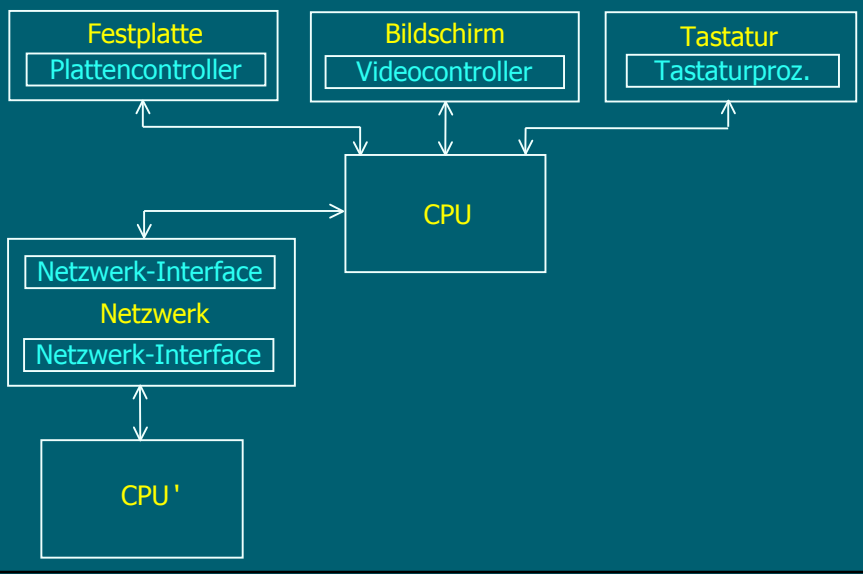
- Laden von Programmen und Daten in Speicher
- Ausgabe von Resultaten auf Bildschirm, Drucker, Festplatte, Diskette...

Ansteuerung

Ansteuerung von Ein-/Ausgabeeinheiten
(= I/O-Einheiten, Peripheriegeräte)
z.B. durch:

- Viedeocontroller (Bildschirm)
- Plattencontroller (Festplatte)
- Tastatur-Prozessoren
- Netzwerkschnittstellen usw.

Ansteuerung (graphisch)



Schnittstellen

Bausteine zum Ansteuern einer I/O-Einheit =
Schnittstellen oder Interfaces

Kontaktaufnahme zwischen CPU und Interfaces

- Von der CPU aus gesehen wie Datenaustausch mit Speicher !
- Der Datenaustausch mit verschiedenen Interfaces erfolgt nach verschiedenen festgelegten Schemata (=Protokollen).
  Dies wird durch Software zum Ansteuern der Interfaces geregelt *(Treiber).*

Vorgehen

- Interfaces, UART, EPROM, Memory Map
- Busprotokolle, Zustandsdiagramme

15.1 Interfaces, UART, EPROM, Memory Map

Bernd Becker – Technische Informatik II

I/O-Ports

Interfaces werden von der CPU wie Speicher mit s Adressen angesprochen.

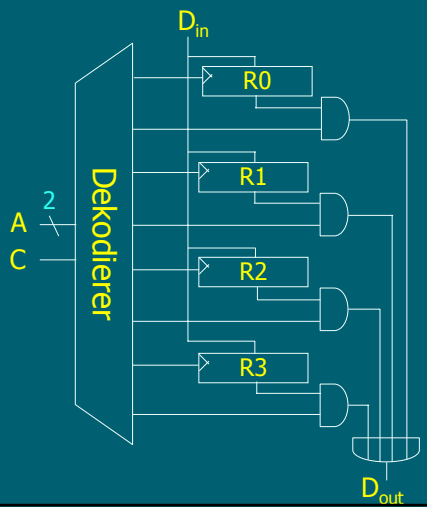
Adressen bei Interface-Einheiten = I/O-Ports

Für Interface mit s I/O-Ports:

- $\lceil \log s \rceil$ Adressleitungen
- /write - Signal

Beispiel:

Interface mit s = 4 Adressen aus CPU-Sicht



4 Register mit
2 Adressleitungen
(Adresse b_{in} (i) für R_i ,
i = 0, 1, 2, 3)

Beispiel (ff)

- R_0: Data in (Peripherie-Einheit schreibt nach R_0, CPU liest)
- R_1: Data out (CPU schreibt nach R_1, Peripherie-Einheit liest)
- R_2: Status (Peripherie-Einheit schreibt Statusinformationen, CPU liest)
- R_3: Command (CPU schreibt Kommandos für Peripherie-Einheit, Peripherie-Einheit liest)

UART

RETI-II mit nur 1 Interface-Baustein:

SIO = Serielle Input/Output-Schnittstelle,

wird zur Verfügung gestellt durch einen speziellen Chip

UART

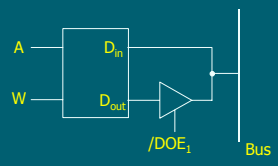
(Universal Asynchronous Receiver and Transmitter)

Eigenschaften der UART

- 8 8 - Bit - Register,
  selektiert durch 3 Adressleitungen
- Wie SRAM mit gemeinsamem Datenein- und –ausgang
  →
  Output enable – Signal /œ
  und
  Schreibsignal /w

Beispiel zur Verwendung

SRAM mit gemeinsamem Dateneingang und –ausgang:



UART XR-16C450: Lesezugriff

Bezeichnung	min	max
Setup-Zeit von A bis /oe	10	
Output enable Zeit		75
Output disable Zeit		50

- Zusatzbedingungen an Timing:
  Lesezykluszeit:
  nach Deaktivieren von /œ dürfen
  /œ und /w mindestens 60 ns nicht aktiviert werden

UART XR-16C450: Schreibzugriff

Bezeichnung	min	max
Setup-Zeit von A bis /w	25	
Hold-Zeit von A nach Ende von /w	5	
Pulsweite	50	
Setup-Zeit von D bis Ende von /w	10	
Hold-Zeit von D nach Ende von /w	25	

Zusatzbedingungen an Timing:

- Schreibzykluszeit:

  Nach Deaktivieren von /w dürfen

  /œ und /w mindestens 85 ns nicht aktiviert werden

EPROM

Nach Einschalten des Rechners soll
ein festes Programm ausgeführt werden,
das über die UART Daten und ein Programm lädt,
das dann ausgeführt wird.

→ festes Programm
in EPROM gespeichert

EPROMs

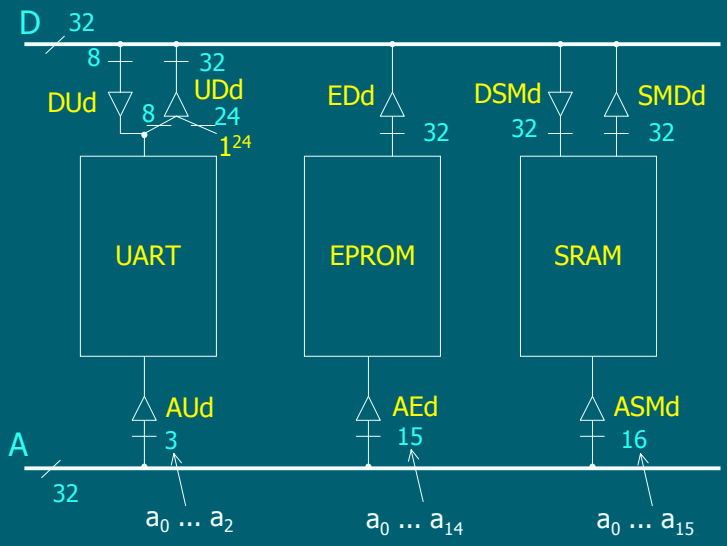
Variante, bei der man die Spezialisierung des OR-Arrays rückgängig machen kann:
erasable PROM = EPROM
EPROMs können durch EPROM-Brenner neu programmiert werden.
Beispiel:
2^{15}x8 EPROM von Hitachi (HN27C256AG-12)
Lesezugriffszeit t_{acc} = (5.0, 120.0)

Datenpfade und Memory Map

Verbinde UART und EPROM mit
Daten- und Adressbus.
(UART mit D[7:0] verbunden)

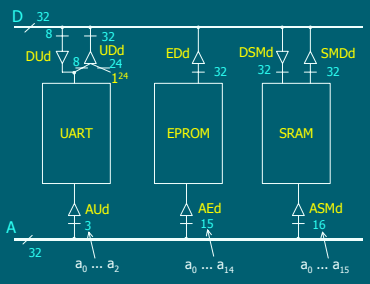
(siehe folgende Abbildung)

Datenpfade (graphisch)



Schalten der Treiber

Datenpfade (graphisch)



- Permanent enabled:

  ASMd, AEd, AUd, DSMd
- Zu erzeugende Signale:
  - Output enable von Treibern SMDd, EDd, UDd, DUd
  - Output enable /Uœ und Schreibsignal /Uw von UART
  - Schreibsignal /SMw von SRAM

Memory Map

Festlegung, welcher Speicher unter welcher Adresse angesprochen wird durch Memory Map.

A[31:30]	Einheit
1 *	SRAM
0 1	UART
0 0	EPROM

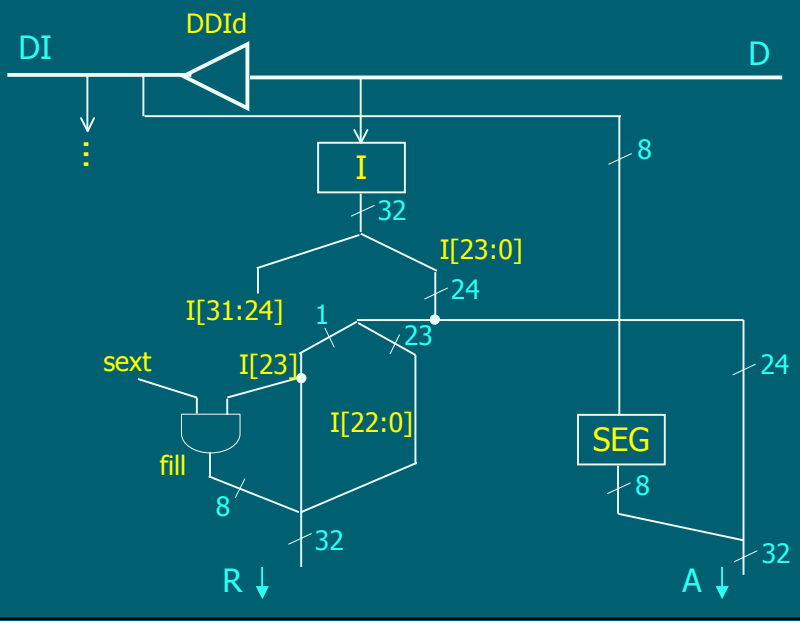
Wenn nach Reset Befehlszähler auf 0 gesetzt wird, wird 1. Befehl aus EPROM geladen !

Problem:

24 - Bit - Konstanten von LOAD, STORE dürfen nicht mehr einfach durch 0^8 ergänzt werden !

Abhilfe:
Segmentregister SEG mit 8 Bit SEG[7:0] !

Schaltung zum Auffüllen der Signale wird verändert:



Zu beachten:

- Bei Segmentwechsel, d.h. Änderung der obersten 8 Bit einer Adresse für LOAD bzw. STORE, muß das Register SEG neu geladen werden !

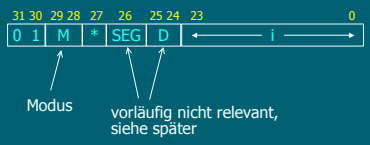
Zu beachten (ff)

- Dazu gibt es 4 neue LOAD-Befehle:
  LOAD SEG i, LOADIN1 SEG i,
  LOADIN2 SEG i, LOADI SEG i
- Zur Kodierung ist I[26] vorgesehen:

31 30	29 28	27	26	25 24	23 ... 0
0 1	M	*	SEG	D	i

Load - Befehle

Prinzip:



Zu beachten (ff)

- Laden in Segmentregister bei I[26] = 1 !
  (Laden in D = ACC jetzt z.B. bei
  $/\,I[26] * I[25] * I[24] = 1$)
- SEG spielt auch eine Rolle,
  wenn SRAM mehr als 24 Adressbits hat !

Noch zu lösende Probleme:

1. EPROM (120 ns) und UART (85 ns) haben viel größere Zugriffszeiten als SRAM (45 ns).
   → Erhöhung der Zykluszeit ??

2. Kontrollsignale für verschiedene *Arten von Speicher* sind zu erzeugen.

zu 1.:

Mache die Länge der Fetch- bzw. Execute-Phase variabel in Abhängigkeit vom Speicher, auf den zugegriffen wird.
schneller Speicher (SRAM) : wie bisher
langsamer Speicher : Einfügen zusätzlicher Zyklen *(Wait-Zyklen)*

zu 2.:

Ziel:

Vermeide komplexe Kontrollogik der CPU.
Speicherzugriffe sollen aus CPU-Sicht uniform behandelt werden.

Lösung:

Auftrennen in mehrere Kontrolleinheiten

zur Lösung:

- eigene Kontrolleinheiten für verschiedene Speicher, hier sind *speicherspezifische* Eigenschaften versteckt (Bsp.: unterschiedliche Zugriffszeiten, spezielle Lese- und Schreibzykluszeit wie z.B. bei UART)
- uniforme Behandlung auf CPU-Seite, Kommunikation mit Speicher über Anlegen von Adressen und Daten und über bestimmte Kontrollsignale

zur Lösung (ff)

- /mreq zum Ansprechen von Speicher
  (Adresse A bestimmt, um welchen Speicher es sich handelt)
- /mw zum Initiieren eines Schreibzugriffs auf A
- Acknowledge-Signal /ack, das die Kontrolle des Speichers an die CPU schickt, um das Einfügen von Wait-Zyklen zu steuern. Solange /ack inaktiv ist, werden noch Wait-Zyklen eingefügt. (genauer: pro Speicher i gibt es ein individuelles Signal $/ack_i$

  $ack = \bigvee ack_i$)