Kapitel 6

Synchronisation und Busvergabe

6.1. Synchronisation

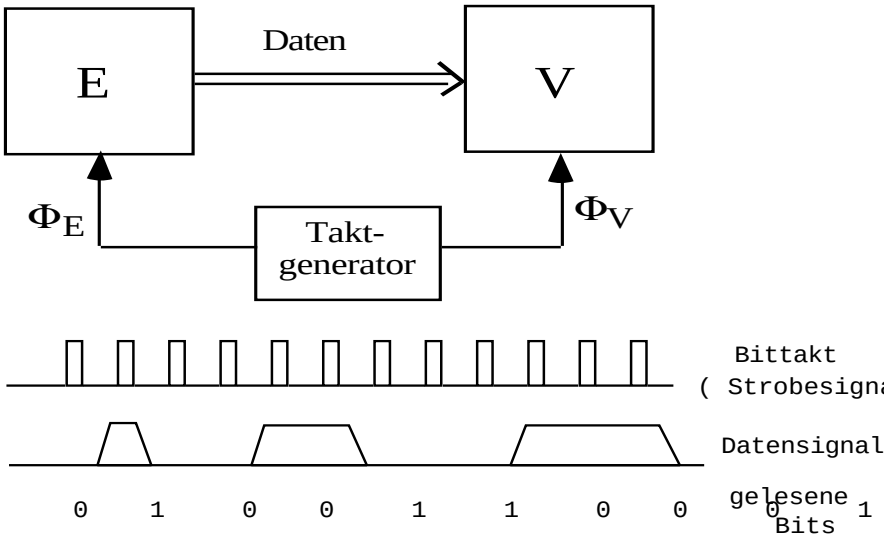
Wenn ein Erzeuger (z. B. die CPU) Daten an einen Verbraucher (z. B. einen Speicher) überträgt, muß der Verbraucher wissen, wann die Daten gültig sind, und der Erzeuger muß unterrichtet werden, wann der Verbraucher die Daten übernommen hat und wann er bereit ist zur Aufnahme der nächsten Daten. Erzeuger und Verbraucher müssen aufeinander synchronisiert werden.

6.1.1. Synchronisation durch gemeinsamen Takt

a) Zentraler Taktgenerator

Ein gemeinsamer Taktgenerator steuert Erzeuger und Verbraucher. (Regelfall innerhalb einer CPU)

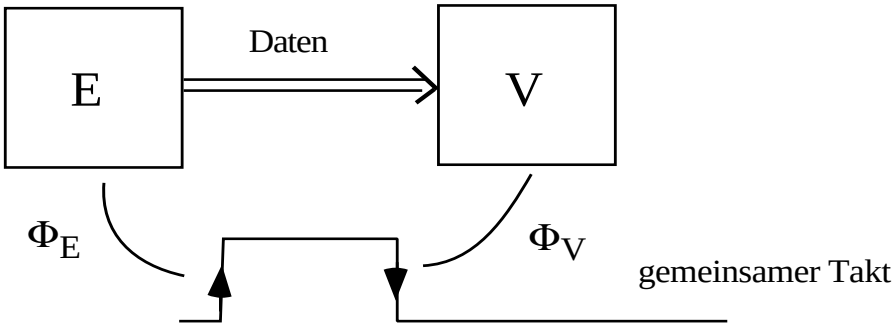
Ein Takt $Ø_E$ sagt dem Erzeuger: "Stelle Daten bereit!" ; ein Takt $Ø_V$ sagt dem Verbraucher: "Übernimm die Daten!". Voraussetzung ist, daß beide, Erzeuger und Verbraucher, schnell sind im Vergleich zur Taktperiode.



Übertragung mit eigenem Bittakt

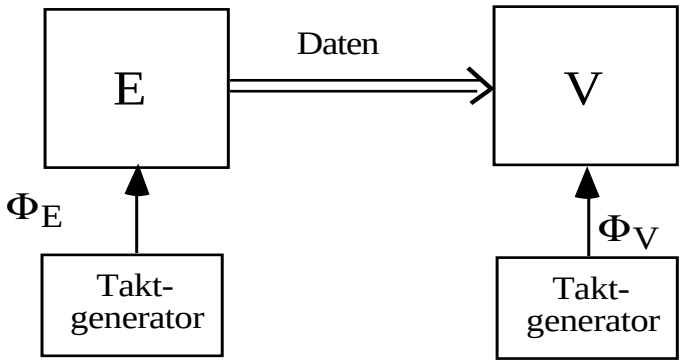
Das Taktsignal $Ø_V$ bezeichnet man auch als **Strobe-Signal**, da es häufig dazu dient, ein Datum in einen Zwischenspeicher einzuschreiben.

Statt zwei Takten $Ø_E$ und $Ø_V$ kann man Vorder- und Rückflanke eines Taktes benutzen.



b) Interne Taktgeneratoren (interne Uhren)

Erzeuger und Verbraucher haben jeweils eigene Taktgeneratoren. Sie müssen anfänglich synchronisiert werden (z. B. durch ein Startsignal) und sollten dann möglichst lange gleichlaufen.



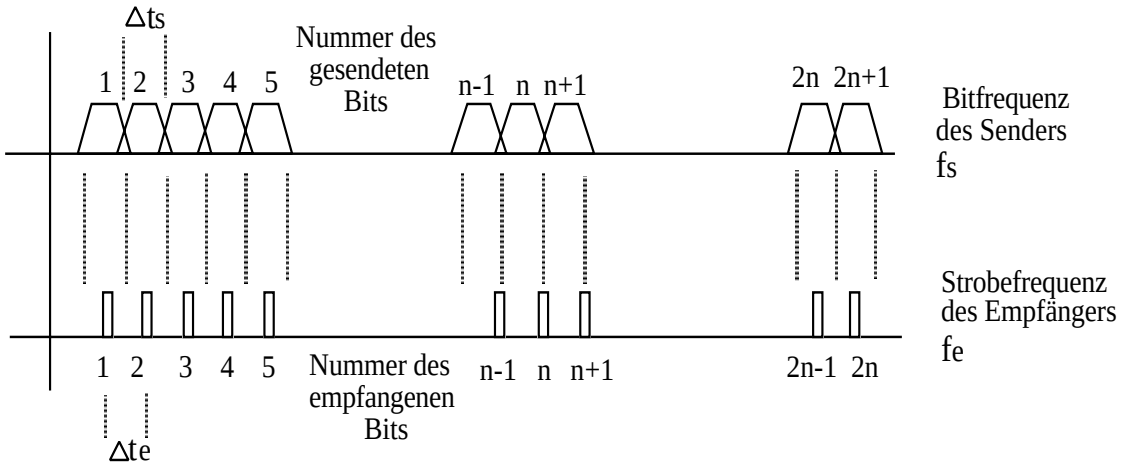
Durch ein Startbit S wird mit einer Verzögerung von T/2 der Oszillator des Verbrauchers gestartet. Sei T die Periode des Oszillators des Erzeugers:

$$T = \frac{1}{f_E}$$

Der Oszillator im Verbraucher hat die Frequenz $f_V = \frac{1}{T_V}$.

Es summiert sich eine Zeitverzögerung Δt auf.

Wenn $\Delta t \approx T/2$ ist, können die Daten nicht mehr zuverlässig gelesen werden.



Sei A die Anzahl der nach dem Startbit übertragenen Nachrichtenbits.

Es ist $\Delta t = A\left(\frac{1}{f_V} - \frac{1}{f_E}\right)$.

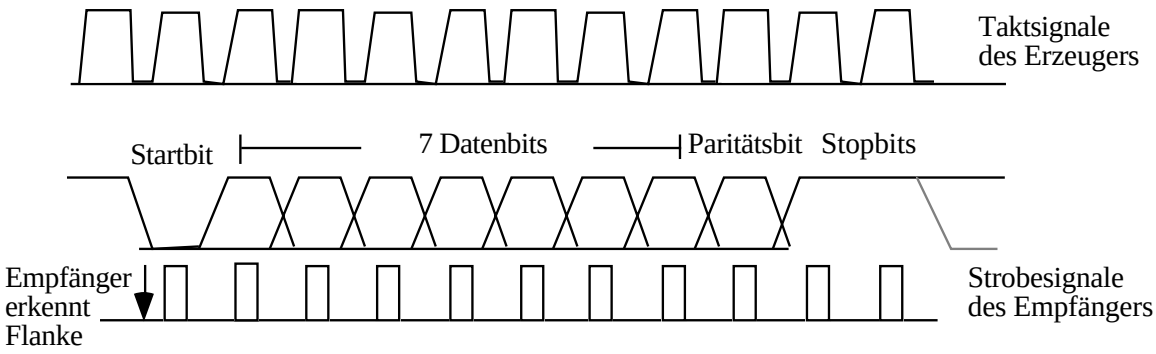
Es wird gefordert $\Delta t \ll \frac{T}{2} = \frac{1}{2f_E}$

$$\Delta t = A \cdot \left(\frac{1}{f_V} - \frac{1}{f_E}\right) \ll \frac{1}{2f_E}$$

$$A \cdot \frac{f_E - f_V}{f_E \cdot f_V} \ll \frac{1}{2f_E} \quad ; \quad f_E - f_V = \Delta f .$$

$$\Rightarrow \quad \boxed{\frac{\Delta f}{f} \ll \frac{1}{2A}}$$

Beispiel: Übertragung von ASCII-Zeichen



Als man diese Codes definierte, war die Synchronisation rein mechanisch:

$A = 10 \Rightarrow \frac{\Delta f}{f} \cdot \frac{1}{20}$

Z. B. mit zwei Oszillatoren, die auf 10^{-2} gleichlaufen, lassen sich ASCII-Zeichen bereits

übertragen.

Heute verwendet man als frequenzbestimmendes Glied in den Oszillatoren Schwingquarze:

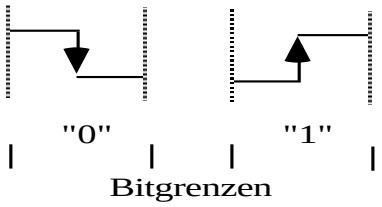
Sie lassen sich problemlos auf $\frac{\Delta f}{f} = 10^{-5}$ einstellen;

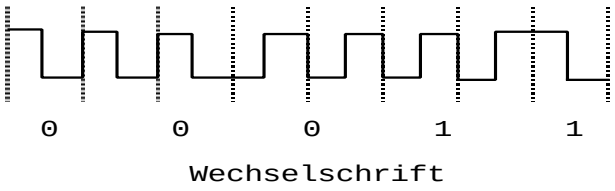
im Prinzip könnten ohne Synchronisationsprobleme 10^4 Bit am Stück übertragen werden.

c) Taktrückgewinnung

Es wird der Takt mit den Daten mit übertragen. Die Bits werden durch Flanken statt durch Pegel codiert: z. B. "0" entspricht einer fallenden, "1" einer steigenden Flanke.

Manchester-Code





Wechselschrift

Mit den Datenflanken kann der Takt rückgewonnen werden.

6.1.2. Synchronisation über eigene Synchronisationsleitungen

Erzeuger und Verbraucher signalisieren über Synchronisationsleitungen, die auf der Gegenseite passiv gelesen werden, den Datentransfer.

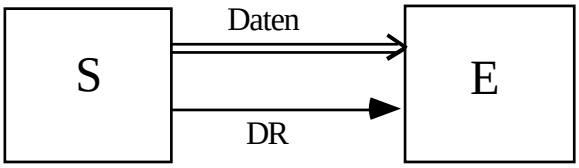
6.1.2.1. Einwegkommandos

Es gibt nur eine Synchronisationsleitung.

a) Quellgesteuert

Der Erzeuger (die Quelle) stellt ein Signal "data ready", DR, zur Verfügung, das dem Verbraucher die Gültigkeit der Daten signalisiert.

Quellgesteuerte Datenübertragung



Der Verbraucher muß schnell sein im Vergleich zum Erzeuger, damit er Daten ohne Verluste abnehmen kann. Mit DR = 1 nimmt er genau ein Datum auf und wartet auf DR = 0; erst dann darf er ein neues Datum lesen.

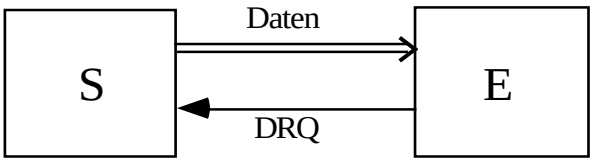
Er muß schnell genug sein, um bei DR = 1 ein Datum lesen und verbrauchen zu können. Der Erzeuger gibt die Datenrate vor.

b) Zielgesteuert

Durch ein Signal "data request", DRQ, fordert der Verbraucher ein Datum vom Erzeuger an.

Der Erzeuger hält das Datum vor, solange DRQ = 1 ist. Der Erzeuger muß schnell sein im Vergleich zum Verbraucher. Der Verbraucher bestimmt die Datenrate.

Zielgesteuerte Datenübertragung

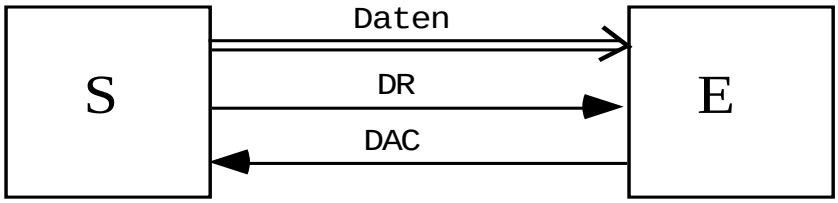


Auch auf ein kurzes Signal DRQ muß der Erzeuger reagieren.

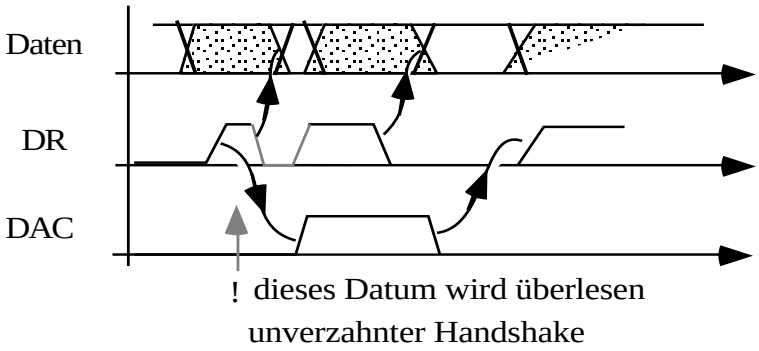
6.1.2.2. Quittungsverfahren

Um von den Verarbeitungsgeschwindigkeiten von Erzeuger und Verbraucher unabhängig zu werden, führt man zwei Synchronisationsleitungen ein:

- DR (data ready) in der Verantwortung des Erzeugers, gelesen vom Verbraucher
- DAC (data accepted) in der Verantwortung des Verbrauchers, passiv gelesen vom Erzeuger.



a) Unverzahntes Quittungsverfahren



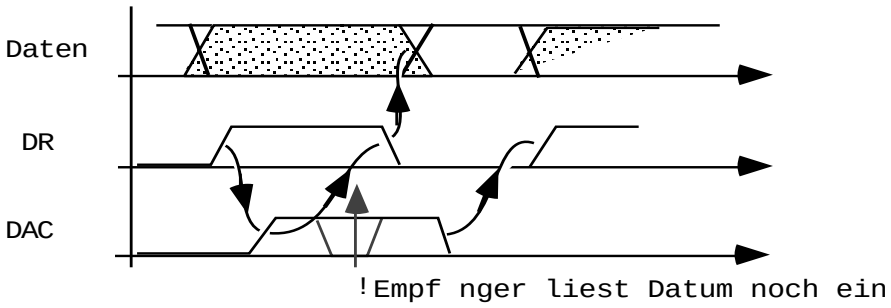
unverzahnter Handshake

Mit DR = 1 kann der Verbraucher Daten übernehmen, mit DAC = 1 meldet er die erfolgte Aufnahme zurück.

Es wird keine Aussage gemacht, wann neue Daten erzeugt werden oder wann der Verbraucher bereit ist zur Aufnahme neuer Daten.

Wenn der Erzeuger schnell ist im Erzeugen der Daten im Vergleich zum Verbraucher, dann wird noch DAC = 0 sein, wenn vom Erzeuger schon wieder DR = 0 gesetzt wird. Der Erzeuger kann dann erneut mit DR = 1 ein neues Datum dem Verbraucher anbieten. Der Verbraucher hat dann das erste Datum überlesen

b) Halbverzahntes Quittungsverfahren



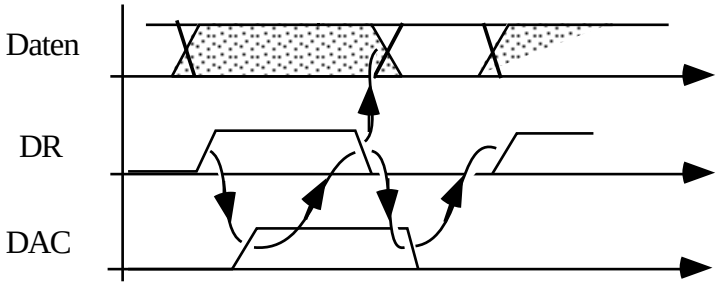
halbverzahnter Handshake

Der Erzeuger beobachtet hier das Quittungssignal DAC.

Erst wenn er DAC = 1 erkennt wird er die Daten durch DR = 0 ungültig schreiben, hat allerdings keine Kontrolle darüber, wann der Verbraucher erneut bereit ist, Daten aufzunehmen.

Setzt der Verbraucher DAC = 0 , wenn noch DR = 1 ist, kann er DR = 1 interpretieren als Aufforderung zum erneuten Lesen eines Datums.

c) Vollverzahntes Quittungsverfahren



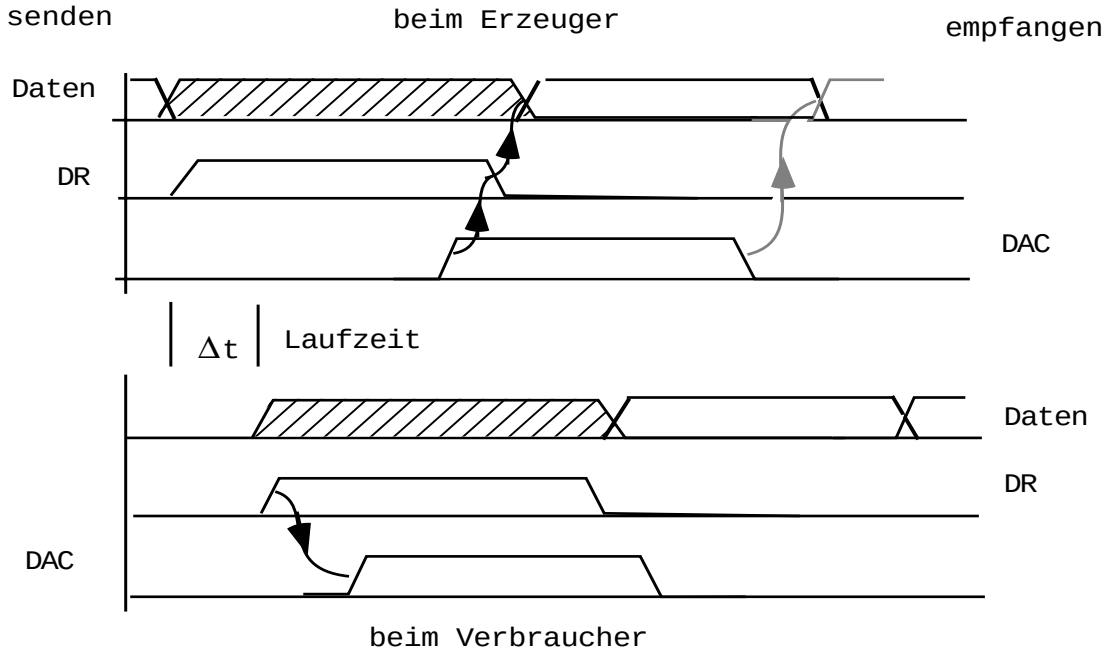
voll verzahnter Handshake

Erst nach DR = 1 kann der Verbraucher Daten übernehmen und quittiert mit DAC = 1.

Dieses Signal veranlaßt den Erzeuger, die Daten ungültig zu schreiben mit DR = 0.

Dies ist Voraussetzung für den Verbraucher, daß er seine Bereitschaft zur Aufnahme neuer Daten mit DAC = 0 meldet.

Das Verfahren arbeitet auch bei beliebig langen Signallaufzeiten zwischen Erzeuger und Verbraucher.



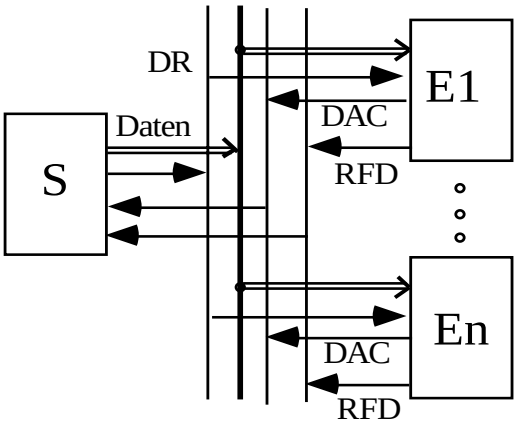
Dreidraht-Handshake

Bei mehreren Verbrauchern, die sich auf einem Sender synchronisieren wollen, sind die Quittungsleitungen, die von den Verbrauchern ausgehen, als wired-and Leitungen zusammengeschaltet: Nur wenn alle Verbraucher die Leitung loslassen, liegt sie auf Eins, sobald ein Verbraucher die Leitung herunterzieht, liegt sie auf Null.

Damit kann die Rückflanke der DAC-Leitung nicht ausgenutzt werden, um die erneute Empfangsbereitschaft der Verbraucher zu signalisieren, sondern es wird eine eigene Leitung ready-for-data (RFD) benötigt. Sie ist genau dann Eins, wenn alle Verbraucher zur

Aufnahme neuer Daten bereit sind und wird heruntergezogen, wenn der erste Verbraucher DR = 1 erkennt.

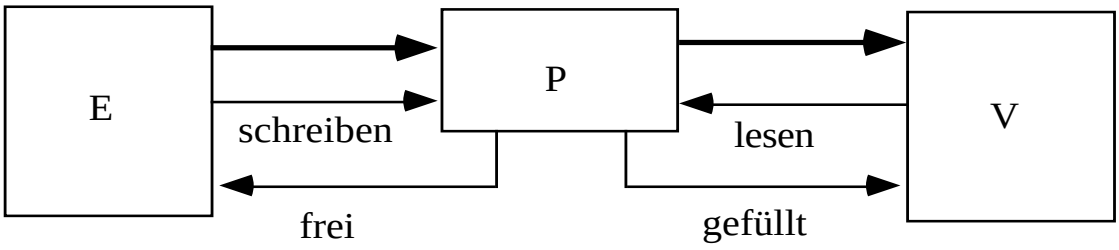
Dreidraht-Handshake



6.1.3. Synchronisation über Zwischenpuffer

Es wird der Verbraucher vom Erzeuger durch ein Zwischenpuffer abgekoppelt. Das macht immer dann Sinn, wenn die Produktionszyklen des Erzeugers unterschiedlich sind von den Verbrauchszyklen.

Der Puffer stellt zwei Signale bereit: "frei" und "gefüllt"; mit "schreiben" transferiert der Erzeuger ein Datum in den Puffer, wenn er "frei" gelesen hat. Mit "lesen" verlangt der Verbraucher ein Datum vom Puffer nachdem ihm "gefüllt = 1" das Vorhandensein des Datums signalisiert hat.



In Form von Pseudocode zeigt das folgende Bild die Aktionen bei Erzeuger, Puffer und Verbraucher.

Erzeuger:

```
repeat
  erzeuge neues Datum;
  wait until FREI;
  schreibe Datum
       in Puffer;
forever
```

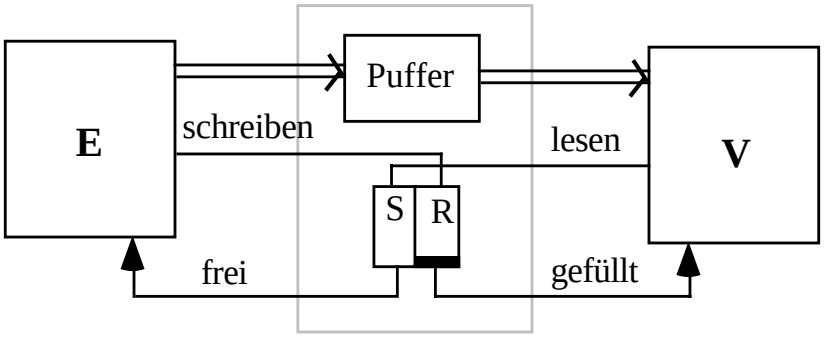
Puffer:

```
repeat
  if FREI then
    wait until SCHREIBEN;
    übernimm Datum ;
    FREI := 0;
    Gefüllt := 1;
  endif
  if  GEFÜLLT then
    wait until LESEN;
    lege Datum heraus;
    GEFÜLLT := 0;
    FREI := 1;
   endif
forever
```

Verbraucher:

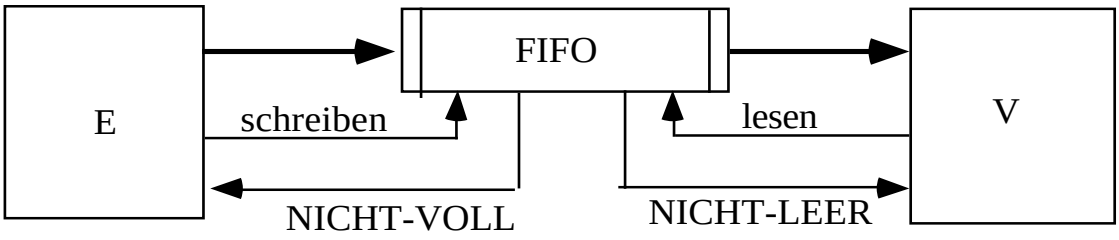
```
repeat
  wait until GEFÜLLT:
  lese Datum aus Puffer;
  verbrauche Datum:
forever
```

Die konkrete Realisierung könnte in Form eines Flip-Flop beim Puffer geschehen.

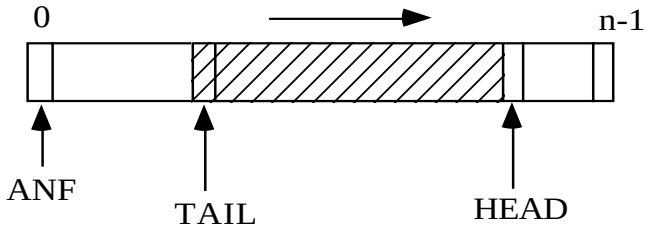


6.1.3.1. Synchronisation über eine Warteschlange

Um unterschiedliche Erzeugungs- und Verbrauchszeiten bei gleicher Rate über längere Zeit hin auszugleichen, schreibt der Erzeuger seine Daten in eine Warteschlange, aus der der Verbraucher nach dem fifo-Prinzip (first-in-first-out) liest.



Technisch wird ein solcher Speicher als Kreisliste realisiert: ein Speicherbereich von n Worten mit zwei Zeigern, HEAD and TAIL.



Die Aktionen am Ort des Erzeugers, des Verbrauchers und im FIFO zeigt die folgende Abbildung.

Erzeuger:

```
repeat

   erzeuge Datum:
   wait until NICHT-VOLL;
   schreibe Datum in FIFO;

forever
```

Verbraucher:

```
repeat

   wait until NICHT-LEER;
   lese ein Datum aus FIFO;
   verbrauche Datum;

forever
```

FIFO

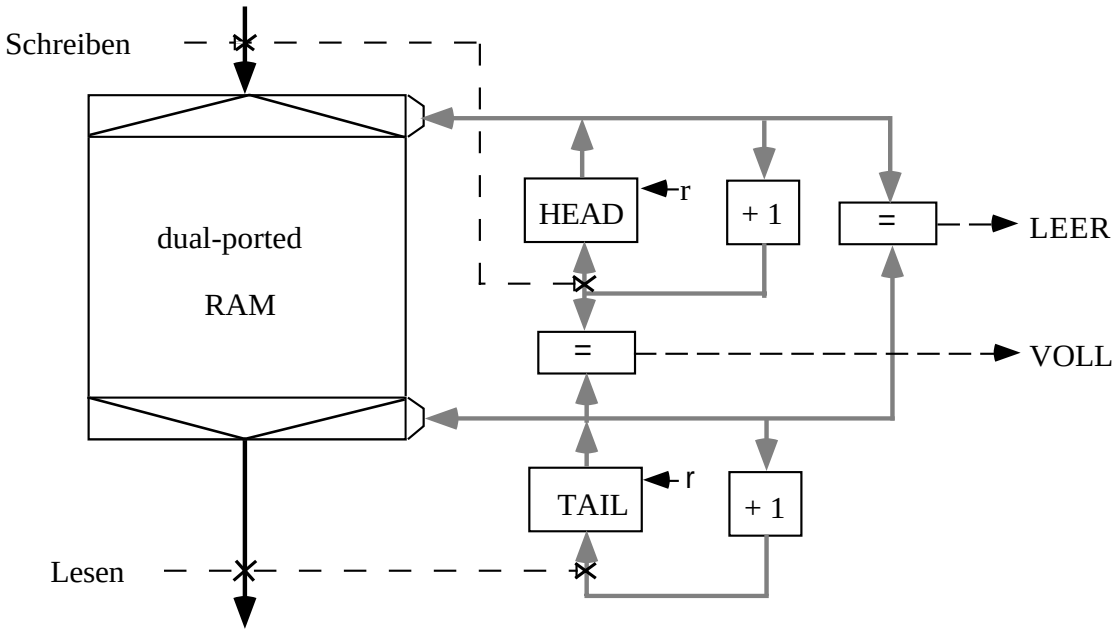
Anfangszustand : HEAD = TAIL = ANF

==> LEER = 1; VOLL = 0

```
:
repeat

   IF SCHREIBEN then
      S(HEAD) := Datum;
      HEAD := (HEAD + 1) mod n;
   endif

   if LESEN then
      Datum := S(TAIL);
      TAIL := (TAIL + 1) mod n;
   endif

   LEER := ( HEAD = TAIL );
   VOLL := (( HEAD +1) mod n) = TAIL )

forever
```

Die Länge der Warteschlange wird bestimmt durch den ungünstigsten Fall: der Verbraucher hat n Daten produziert, die der Verbraucher noch nicht abgenommen hat. Wenn die mittlere Produktionsrate des Erzeugers größer ist als die mittlere Abnahmerate des verbrauchers, dann wächst n über alle Grenzen.

Die Realisierung kann durch ein dual-ported memory mit angeschlossenen Zeigern geschehen.



Die direkte Realisierung eines FIFO ist weniger gebräuchlich; man müßte Schieberegister gebrauchen, um ein Gebilde nachzubauen, das sich wie ein Zigarettenautomat verhält: von "unten" her werden Daten entnommen und vorhandene Daten "rutschen" nach.

6.1.3.2. Synchronisation von Ein/Ausgabe

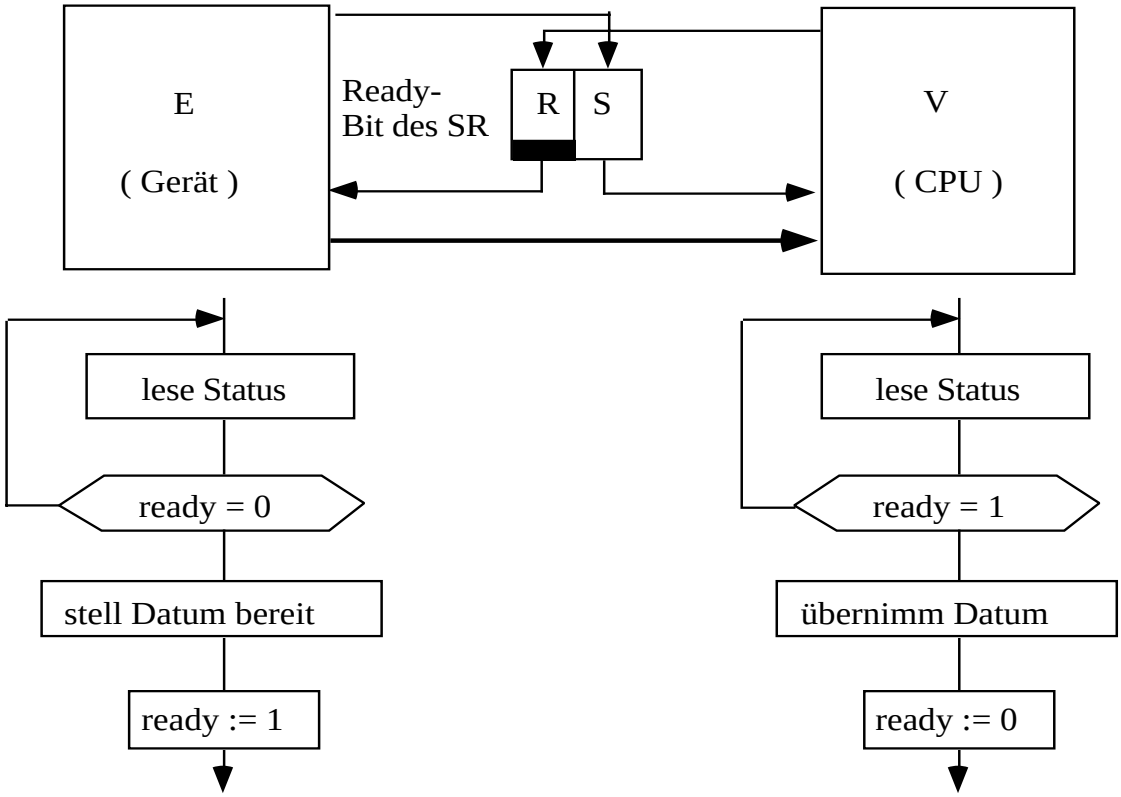
Die Synchronisation zwischen CPU und externen Geräten geschieht über ein Bit (Flip-Flop) im Statusregister des Interface des Gerätes.

a) Das externe Gerät sei langsam gegen die CPU

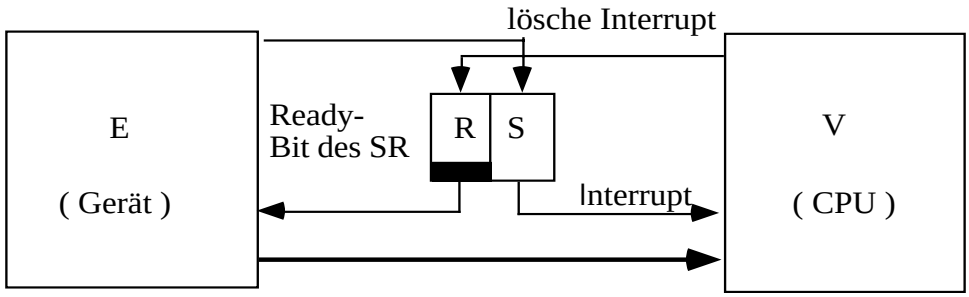
Wenn das Gerät sich vergewissert hat, daß die CPU das vorherige Datum abgenommen hat, stellt sie das nächste Datum bereit und meldet mit READY = 1.

Die CPU kann auf zweierlei Weise reagieren:

- Sie pollt fortlaufend das Ready-Bit (busy waiting)



- oder das Setzen des Ready-Bit erzeugt einen Interrupt in der CPU.



Dieser letztere Fall ist immer dann sinnvoll, wenn es sich bei dem Gerät um Eingabegeräte handelt, bei denen am anderen Ende ein Mensch sitzt.

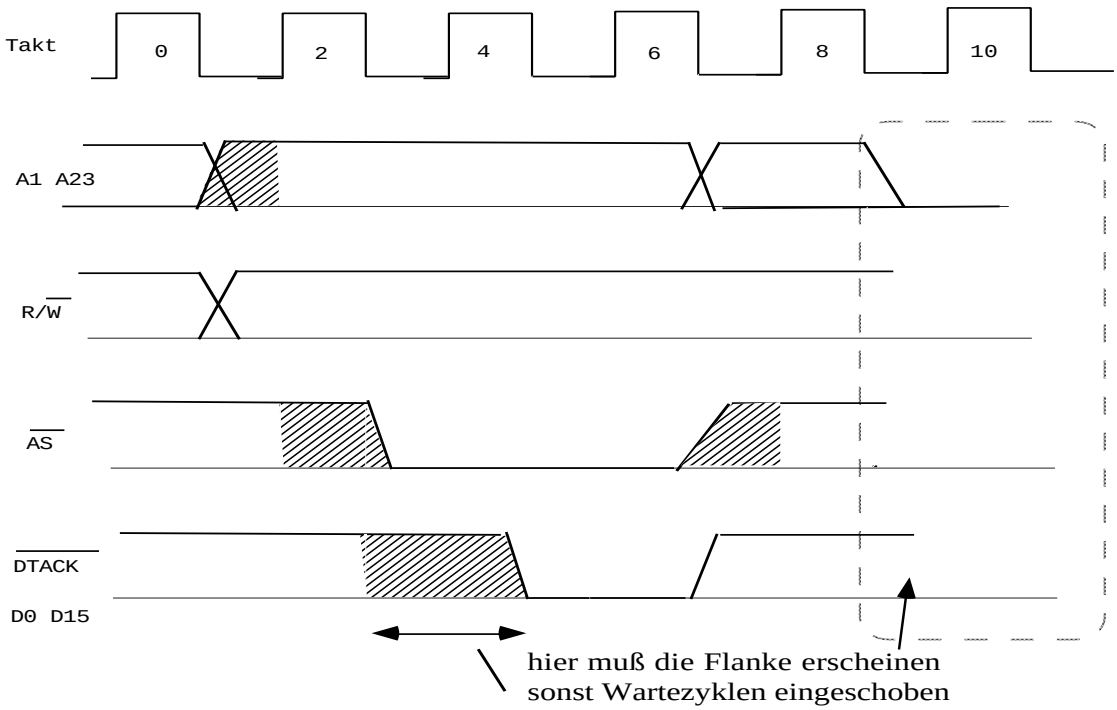
Er produziert alle 100 ms ein Zeichen oder rutscht die Maus etwas weiter; für den Rechner sehr lange Zeiten, in denen er 10^5 - 10^6 Befehle abarbeiten kann.

b) Das externe Gerät ist vergleichbar schnell wie der Rechner

In diesem Fall macht busy-waiting Sinn: Der Rechner schreibt gegebenenfalls einige Wartezyklen ein, wenn das externe Gerät nicht schnell genug ist.

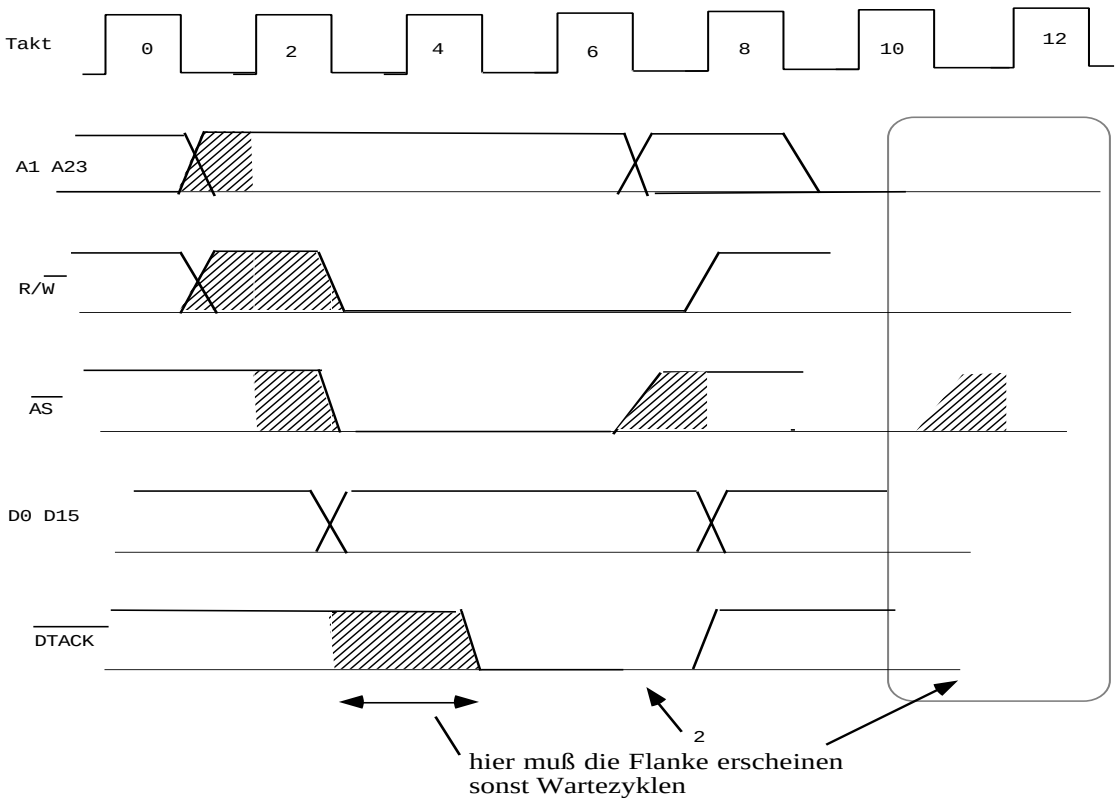
Beispiel: Schreib-Lesezyklen beim MC 680X0 - Rechner

Ablauf eines Lesezyklus aus dem Speicher



Die schraffierten Flächen geben die erlaubten Zeiten zum Setzen der Signale an.

Kommt das DTACK-Signal (data-transfer-acknowledge) vom Speicher nicht rechtzeitig, werden Wartezyklen eingeschoben (wait-states).



Ablauf eines Schreibzyklus

Ähnlich ist das Vorgehen beim Schreiben in den Speicher. Wenn der Speicher die Übernahme der Daten mit DTACK nicht rechtzeitig quittiert, schiebt die CPU Wartezyklen ein.

In vielen Fällen sind externe Speicher vom Fleck weg langsamer als die CPU: Dann kostet jeder Zugriff ein oder mehrere Wartezyklen (wait-states).

6.2. Busvergabe

Es geht hier um die allgemeine Frage, wie der Zugriff auf ein Betriebsmittel organisiert ist, auf das im Prinzip gleichzeitig mehrere Benutzer zugreifen können, das aber nur von 1 bis n Benutzern gleichzeitig gebraucht werden kann.

Die Frage wird verengt auf das spezielle Betriebsmittel "Bus":
Zu einer Zeit darf nur genau einer schreiben, aber viele dürfen parallel die Daten auf dem Bus lesen.

Die notwendige Schiedsrichterfunktion, die den Konfliktfall lösen muß, daß mehr als einer gleichzeitig schreibend auf den Bus zugreifen will (z. B. CPU und DMA), kann in Hard- oder Software realisiert werden.

Die Realisierung in Software sind die Protokolle bei Nachrichtentransportsystemen, z. B. Ethernet (s. dort).

Hier soll der Fall betrachtet werden, daß eigene Busvergabeleitungen zur Verfügung stehen.

Um den Zugriffskonflikt zu lösen, muß jedem Teilnehmer eine Priorität zugeordnet werden, da ggf. zwei Teilnehmer "gleichzeitig" d.h. innerhalb eines beliebig kleinen Zeitintervalls, auf den Bus zugreifen wollen.

6.2.1. Hardwaregestützte Busvergabe

Man unterscheidet nach dem Sitz der Schiedsrichterfunktion zwischen zentraler und dezentraler Busvergabe.

Bei der zentralen Busvergabe gibt es eine einzige Busvergabeeinheit während bei der dezentralen Busvergabe die Schiedsrichterfunktion bei jedem Teilnehmer implementiert ist.

Die notwendige Priorität kann fest eingestellt sein (feste Priorität) oder dynamisch vergeben werden (round robin).

Feste Priorität : Seien T_1 ... T_n die Teilnehmer mit den Prioritäten p_1 bis p_n.

Sei $p_i \neq p_j \; \forall \; T_i, T_j$

Bei der Vergabe erhält Teilnehmer T_i den Bus, wenn $p_i > p_j$ für alle Mitbewerber T_j ist.

Um Teilnehmer mit niedrigen Zugriffsraten nicht "verhungern" zu lassen, erhalten sie eine höhere Priorität als Teilnehmer mit häufigem Zugriff.

Dynamische Priorität : Seien T_1 bis T_n Teilnehmer und T_i habe das Betriebsmittel. Dann wird die Priorität neu berechnet: T_i erhält Priorität $p_i = 0$, T_{i+1} Priorität $p_{i+1} = n - 1$, T_{i+2} die Priorität $p_{i+2} = n - 2$, ...

Das ist ein im Mittel faires Verfahren.

Verfahren der Busvergabe

Man unterscheidet

- verkettete Busvergabe (daisy chaining)
- Anrufsuche (polling)
- unabhängige Vergabe (independent request)

Die Vergabeverfahren lassen sich mit der Organisationsform und der Prioritätssteuerung beliebig kombinieren.

In der Tabelle sind die häufigsten Kombinationen durch einen Stern markiert.

Busvergabe

	Priorit t	verkettet	Anrufsuche	unabh ngig
zentral	fest	*		
	round robin		*	
dezentral	fest			
	round robin	*		

Nun zu den einzelnen Verfahren

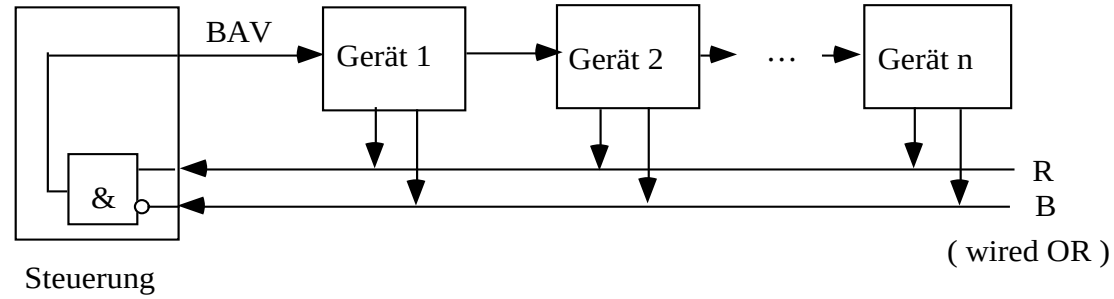
6.2.1.1. Zentrale verkettete Busvergabe (daisy chaining)

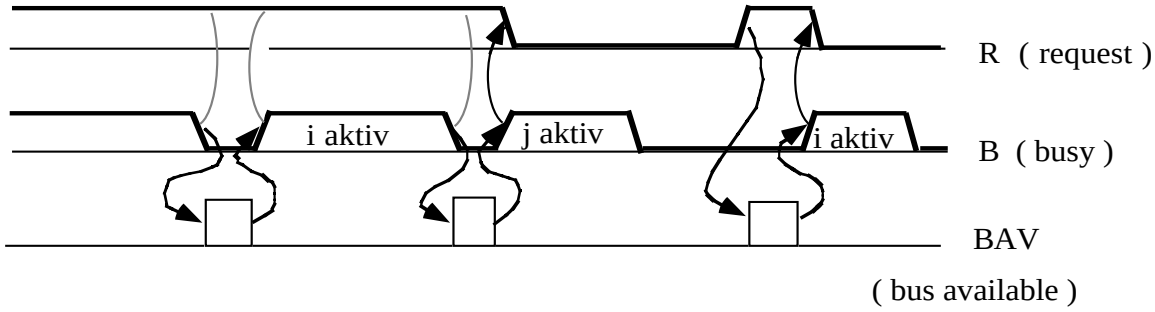
Eine spezielle Freigabeleitung (bus available, BAV) wird durch die Geräte hindurchgeschleift (daisy chaining).

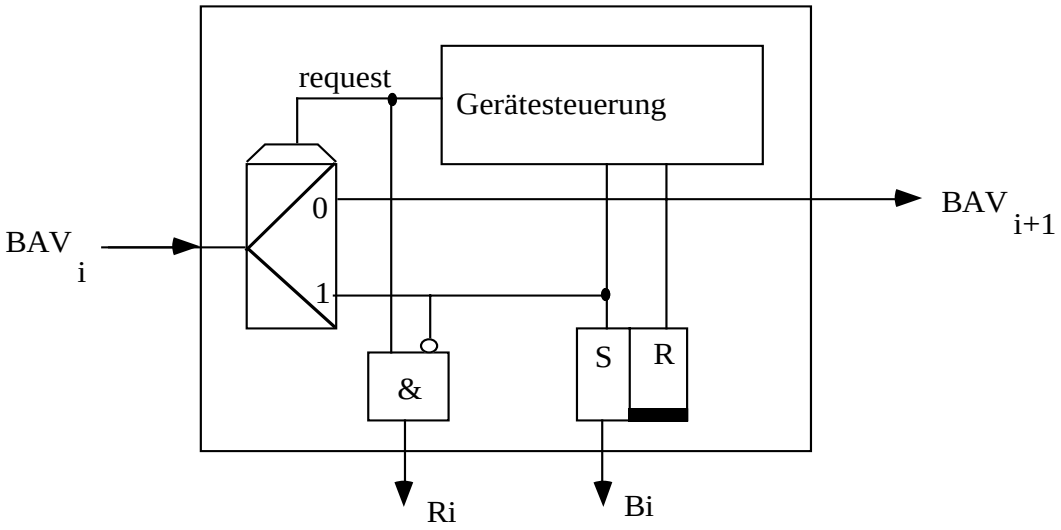
Ein Gerät, das den Bus haben will und diesen Wunsch durch ein Signal auf einer Leitung REQUEST signalisiert hat, wartet auf das Signal BAV und leitet es nicht weiter, sondern requiriert damit den Bus über ein Signal BUSY. Geräte, die den Zugriff nicht wollen, leiten das BAV-Signal weiter.

Die Priorität ist gegeben durch die Lage des Gerätes in Bezug auf die Erzeugung des BAV-Signals.

Verkettete Busvergabe mit zentraler Vergabesteuerung und fester Priorität findet man sehr häufig in Rechnern: Die Geräte am Zentralbus werden so verwaltet.







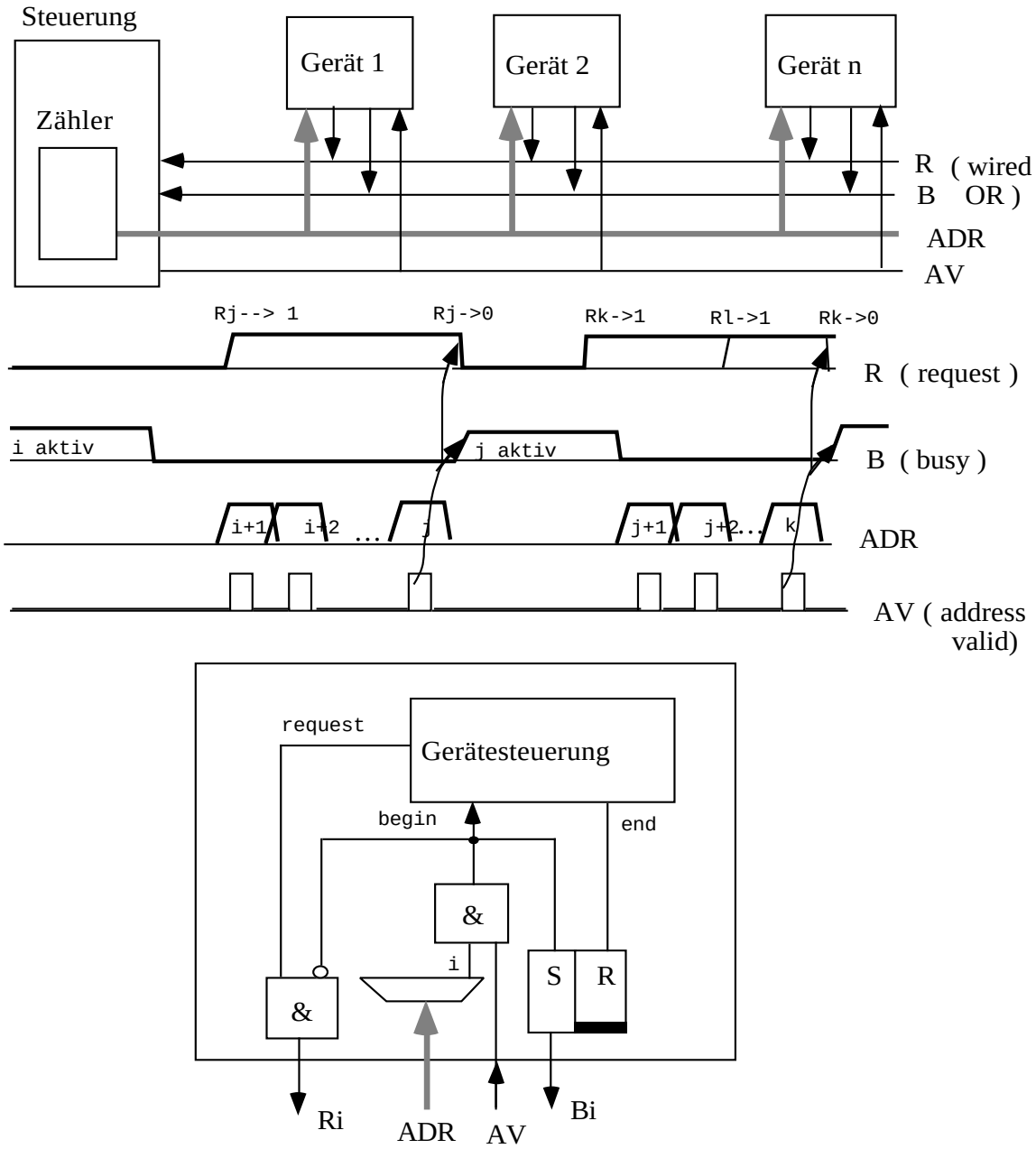
Eine mögliche Realisierung der Weiterleitung der Busvergabe im Gerät ist ebenfalls gezeigt.

6.2.1.2. Zentrale Busvergabe durch Anrufsuche (polling)

Die einzelnen Geräte werden nacheinander abgefragt. Das Gerät, das einen Belegungswunsch hat, belegt mit BUSY den Bus, wenn es seine Adresse erkennt. Die zentrale Busvergabesteuerung erzeugt nacheinander die Adressen der Geräte und signalisiert ihre Gültigkeit mit "address valid, AV".

Mit diesem Verfahren lassen sich sowohl feste als auch dynamische Priorität der Geräte realisieren: die Vergabeeinheit erzeugt nacheinander alle Adressen beim Erkennen von REQUEST = 1 und BUSY = 0 ; das Gerät mit der ersten Adresse hat höchste Priorität. Genausogut kann "round robin" erzeugt werden: Wenn ein Gerät den Bus erhält, setzt die

Vergabeeinheit die Adresserzeugung beim nächsten Erkennen von "REQUEST = 1 und BUSY = 0" mit der nächsten Adresse fort. Damit erhält das Gerät, das eben den Bus hatte, die niedrigste Priorität.



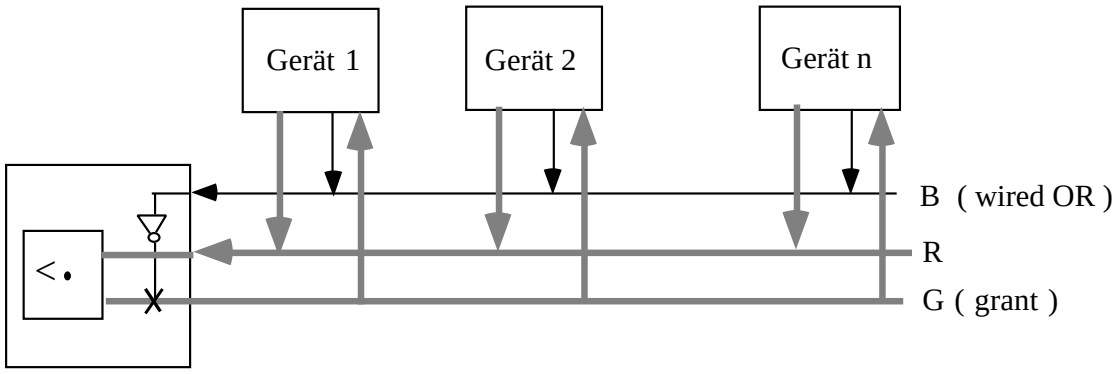
Die Adresserzeugung geschieht durch einen Zähler mod N , wenn N Geräte am Bus hängen.

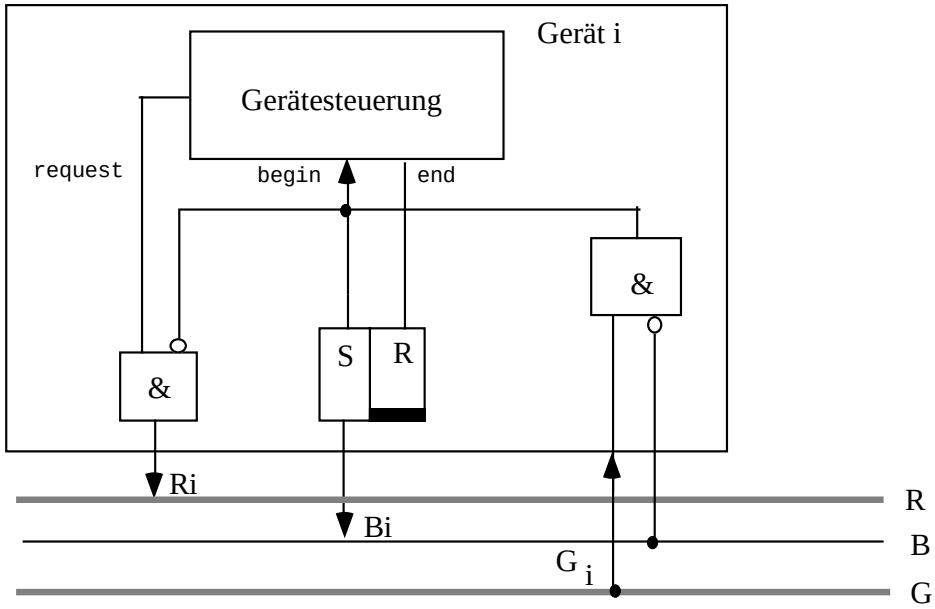
Diese Art der Busvergabe ist z. B. im IEC-Bus realisiert, der in der Prozeßdatenverarbeitung eingesetzt wird, um Meßgeräte an Rechner anzuschließen.

6.2.1.3. Zentrale Busvergabe mit unabhängigem Zugriff (independent request)

Jedes Gerät hat eine eigene Stichleitung R (request) zur und G (grant) von der Busvergabesteuerung.

Dort wird das Gerät bei fester Priorität mit $p(R_i) = \max(p_i)$ ausgewählt, und mit G_i diesem Gerät der Bus gegeben. Das Gerät belegt daraufhin die Belegtleitung B.

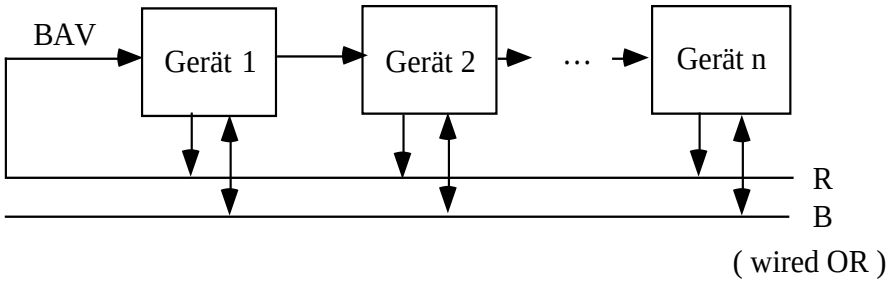


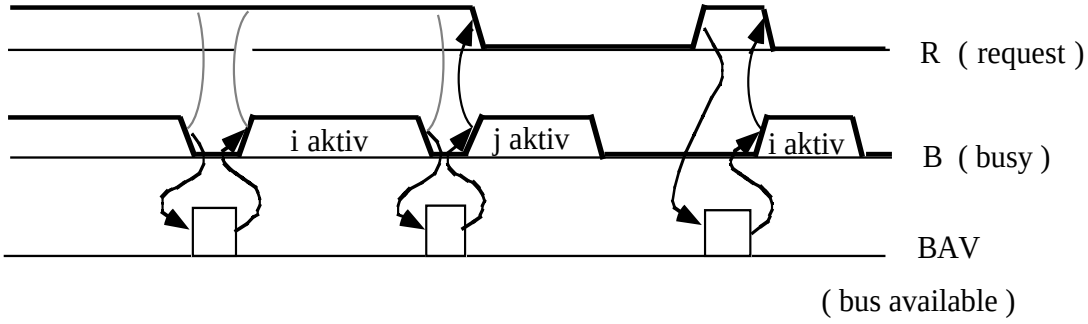


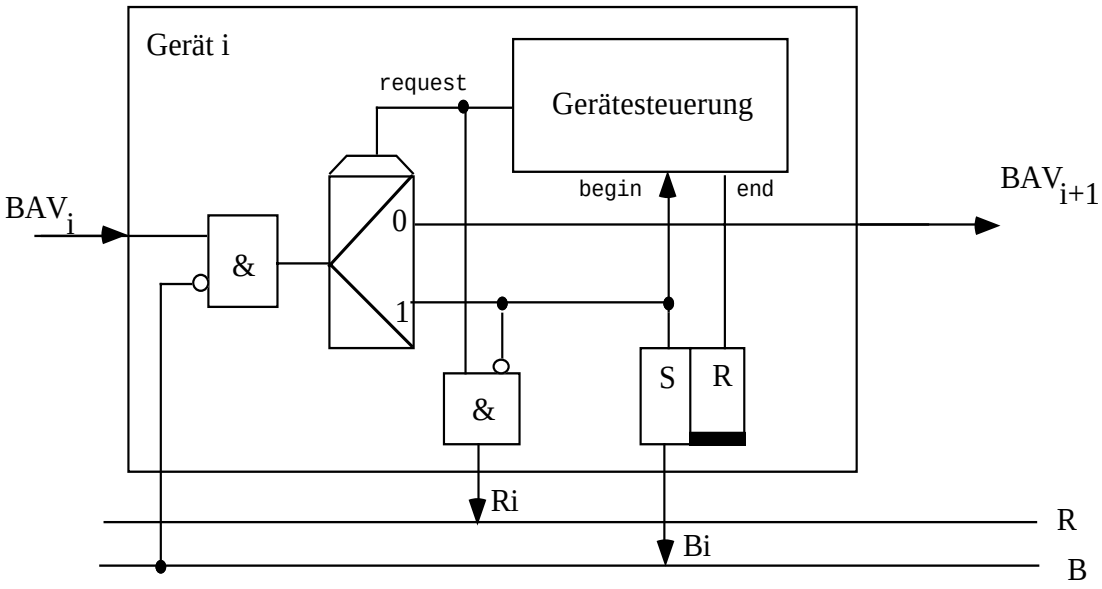
6.2.1.4. Dezentrale Busvergabe mit Verkettung (daisy chaining)

Verlegt man die Erzeugung des Freigabesignals in die Geräte hinein, kann man ohne zentrale Steuerung auskommen. Das Gerät, das als letztes den Bus belegte, übernimmt dann die Rolle der Busvergabesteuerung für den nächsten Zugriff auf den Bus.

Die Realisierung ist besonders einfach für feste Priorität

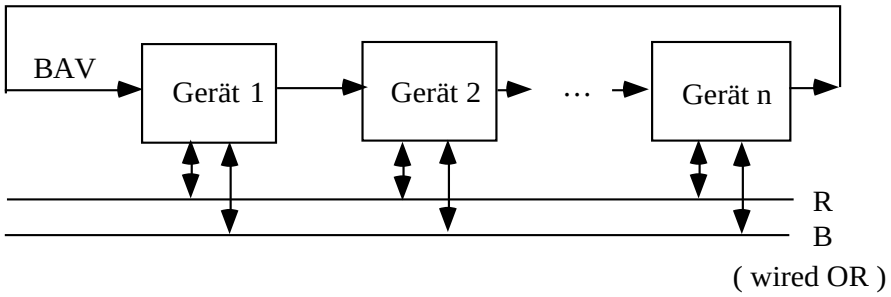


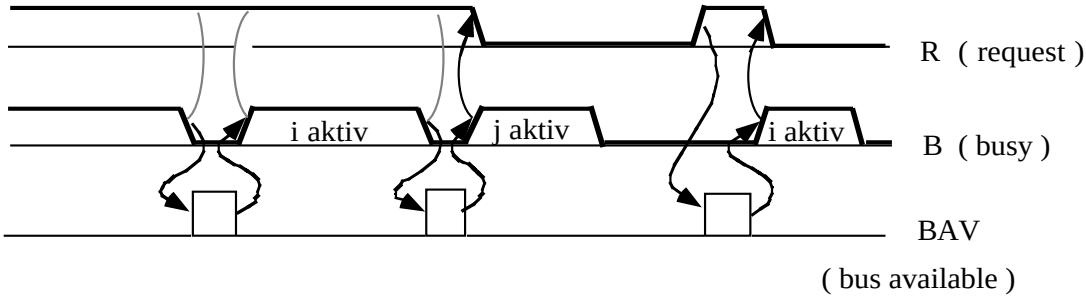


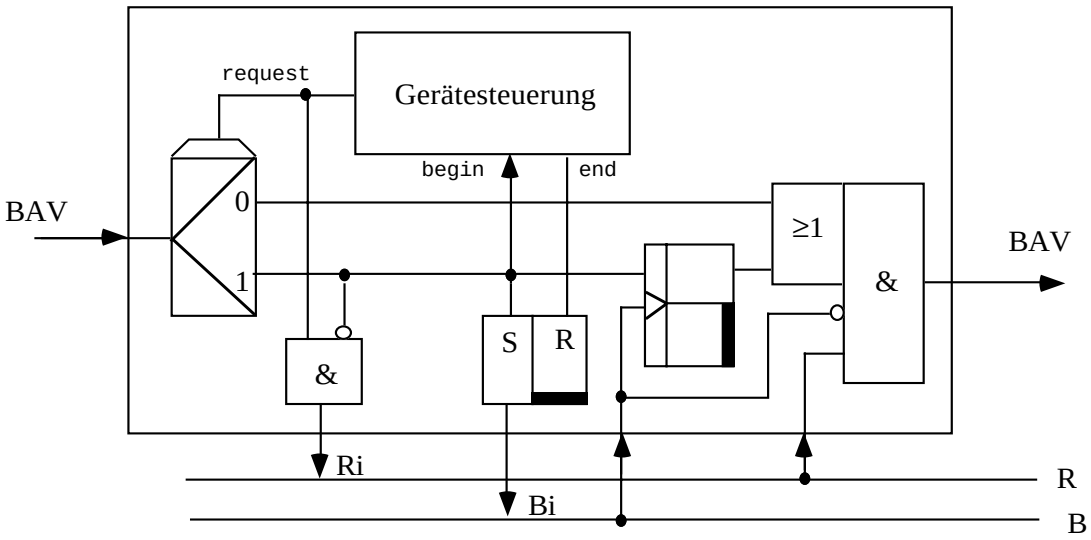


das Signal BAV wird automatisch erzeugt aus R und in das erste Gerät eingeschleift.

Auch dynamische Priorität läßt sich realisieren

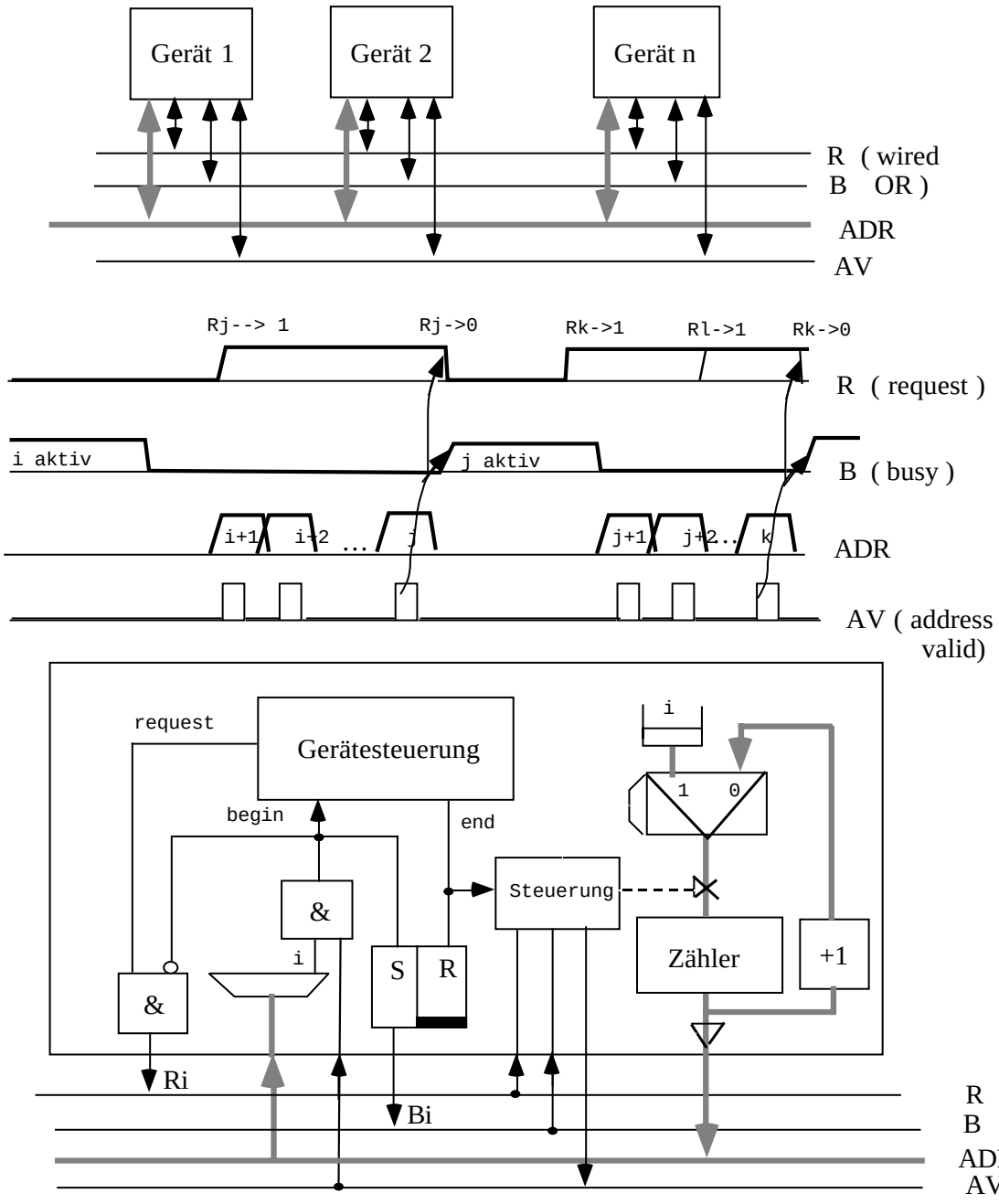






hier erzeugt das Gerät, das als letztes den Bus hatte, das Signal BAV wenn R = 1 und B = 0 gilt.

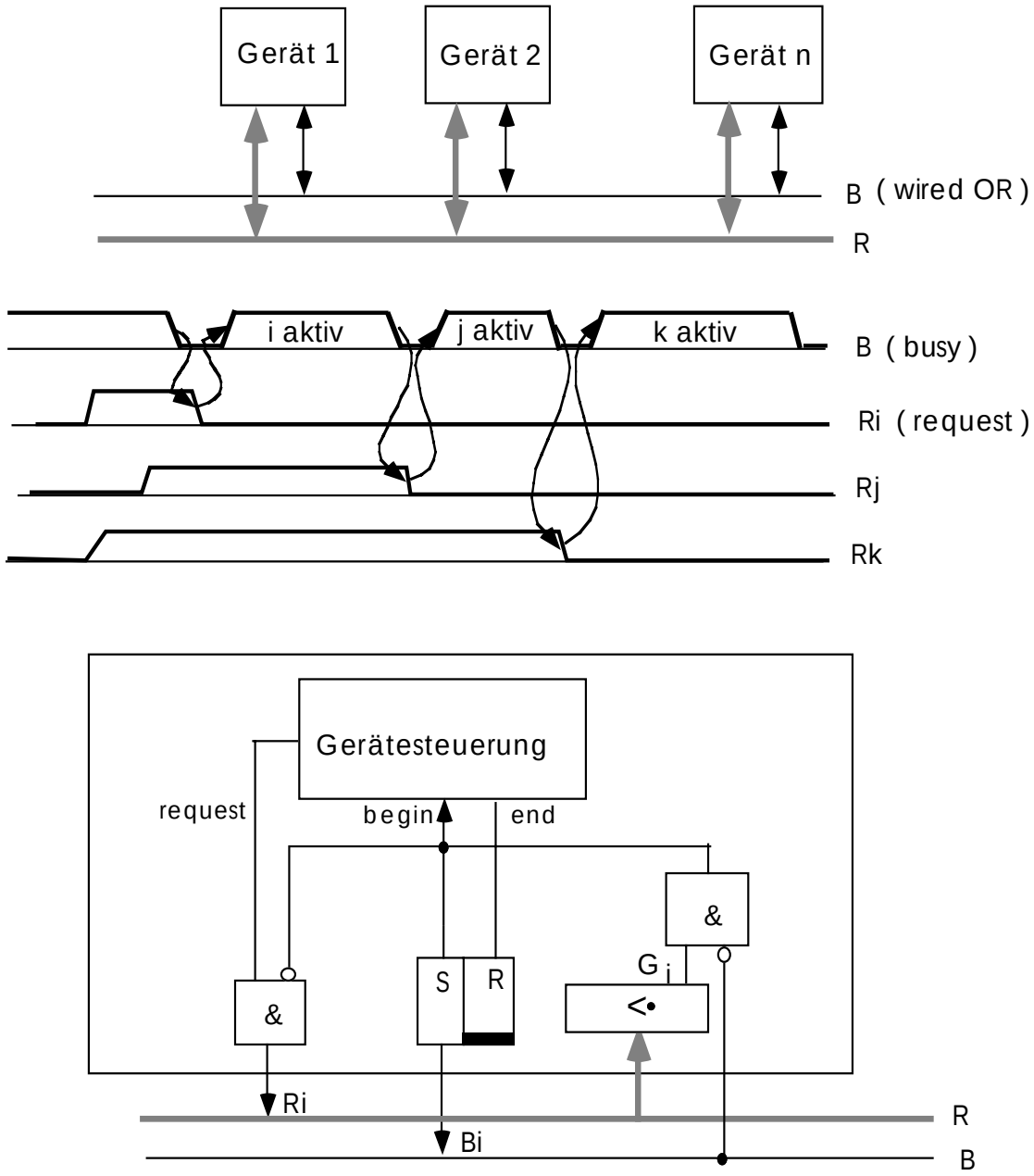
6.2.1.5. Dezentrale Busvergabe mit Anrufsuche (polling)



In jedem Gerät gibt es einen Zähler mod N , der die Abfrageadressen erzeugen kann.

Das Gerät, das als letztes den Bus hatte, erzeugt beim nächsten Zugriff die Abfrageadressen.

6.2.1.6. Dezentrale Busvergabe mit unabhängigem Zugriff (independent request)



Ein Gerät, das den Bus haben möchte, setzt sein Requestbit Ri = 1, und beobachtet das anliegende Anforderungspattern (R1,…, Rn). Nur wenn seine Anforderung Ri die höchste Priorität hat und der Bus nicht belegt ist, darf es den Bus durch Setzen von B für sich requirieren.

6.2.2. Semaphore und Monitor

Wenn der Zugriff auf ein gemeinsames Betriebsmittel nicht durch Vergabeleitungen zu regeln ist, muß der Zugriff durch einen eigenen Mechanismus bei dem Betriebsmittel geregelt werden.

Im Falle des Zugriffs durch mehrere Prozesse, die parallel zueinander laufen, aber von denen zu einer gegebenen Zeit nur einer jeweils aktiv ist, kann der Zugriff durch eine Semaphore oder einen Monitor geschehen.

Semaphore

An das Betriebsmittel wird eine Variable S gebunden, auf der zwei Zugriffsfunktionen P(S) (passer) und V(S) (verlaat) arbeiten.

```
Vor Zugriff auf das Betriebsmittel:
P(S):    if S = 0 then wait
                  else S := S-1;
          Zugriff auf das Betriebsmittel

Vor Rückgabe des Betriebsmittels
V(S):    S := S + 1
          Rückgabe des Betriebsmittels
```

Die kritische Operation ist P(S); wenn Interrupts auftreten, könnte zwischen der Abfrage auf S = 0 und dem Dekrementieren S := S - 1 ein zweiter Prozeß das gleiche Betriebsmittel requirieren wollen. Dann sichert sich der erste Prozeß anschließend das Betriebsmittel, obwohl es zwischenzeitlich vergeben wurde.

Es muß also P(S) unteilbar oder **atomar** sein.

- Es gibt einen Maschinenbefehl "teste Flag F und falls F = 1 setze F := 0" , der nicht unterbrechbar ist.
- P(S) sperrt zunächst alle Interrupts und führt dann nacheinander aus: "teste Flag F" , "falls F = 1 setze F := 0" und gibt dann die Interrupts wieder frei.

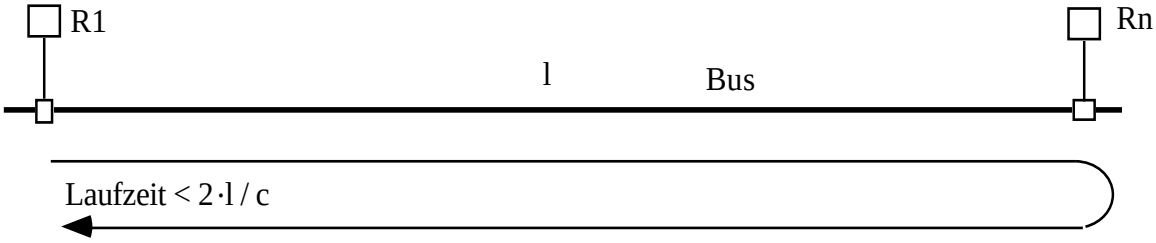
Die zweite Möglichkeit ist die Bindung des Betriebsmittels an einen Monitor, der als "Platzwart" fungiert. Er stellt Zugriffsprozeduren zur Verfügung, die dann gesichert (d. h. intern über eine Semaphore) auf das Betriebsmittel zugreifen.

6.2.3. Vergabe des Zugriffs auf serielle Kommunikationsverbindungen

6.2.3.1. CSMA/CD

Bevor ein Teilnehmer auf den Verbindungskanal zugreift, vergewissert er sich, daß dort kein Verkehr ist (carrier sense, CS).

Dann schickt er eine Präambel und hört mit, ob ein anderer ebenfalls sendet (collision detection, CD). Gegebenenfalls wird der Sendeversuch abgebrochen und nach einer statistischen Wartezeit erneut versucht (multiple access, MA).

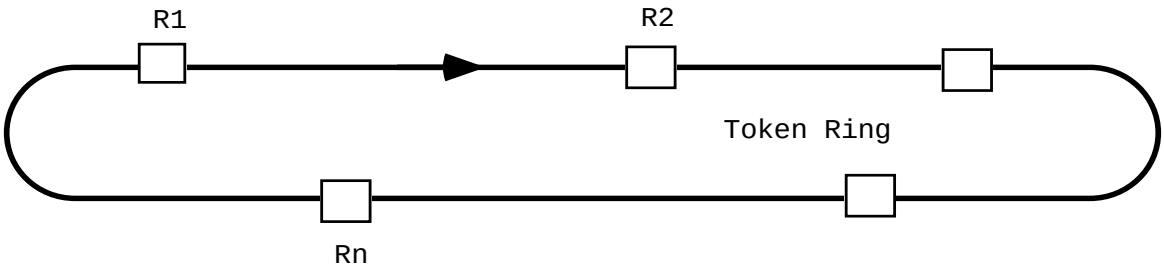


Dieses Verfahren ist brauchbar für viele Teilnehmer an einem Bus oder einer gemeinsam genutzten Funkstrecke. Nach einer Zeit $T > 2 \cdot l / c$ mit c als Ausbreitungsgeschwindigkeit des Signals auf der Leitung - typ. c = 200 m/µs - weiß ein Teilnehmer, ob eine Kollision stattgefunden hat. Dann kann er seine Nutzdaten senden, ohne daß er unterbrochen werden kann. Zugriffe auf Ethernet-Verbindungen werden so gestaltet.

6.2.3.2. Token Ring

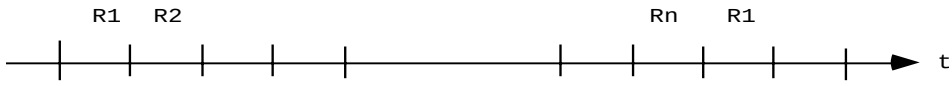
Wenn die Teilnehmer topologisch zu einem Ring zusammengeschlossen sind, kann dort eine Sendeerlaubnis (token) kreisen: Wer an der Kommunikation teilnehmen will, wartet bis das Token bei ihm ankommt, nimmt es fort, sendet seine Nachricht und gibt es dann erst weiter. Ein Teilnehmer, der nicht senden will, gibt das Token direkt weiter.

Probleme entstehen bei langen Nachrichten: ist das Token verlorengegangen oder wird nur excessiv lange gesendet?



6.2.3.3. Slotted Ring (Zeitschlitzverfahren)

Hier erhält jeder Teilnehmer einen Zeitschlitz fester Länge zugeteilt, in dem er senden darf.



Das begrenzt die mögliche Länge von Nachrichten, und beschränkt zugleich die Ausnutzung der verfügbaren Zeit, denn ein Teilnehmer erhält seinen Zeitschlitz, auch wenn er nicht senden will und kein anderer darf in dieser Zeit senden.