Speichertechnologien (memory)

Einleitung

- Anfang/Mitte der 70er-Jahre wurden, die in der Rechenertechnik damals hauptsächlich verwendeten magnetischen Speicher (Ferritkernspeicher, Magnetbandspeicher etc.) und mechanischen Speicher (Lochband- oder Lochkartenspeicher etc.) durch digitale Halbleiterspeicher ersetzt
- Hauptmotivation war wesentlich kürzere Zugriffszeit, geringeres Bauvolumen, höherer Zuverlässigkeit und größere Speicherkapazität zu erreichen
- Digitale Halbleiterspeicher speichern ähnlich wie FFs pro Speicherzelle ein einzelnes Bit
- Digitale Halbleiterspeicher erreichen durch Gruppierung und hoch entwickelte Fertigungstechnologien sehr viel höhere Speicherdichten als FFs
- Zugriffszeiten von digitale Halbleiterspeichern sind aber langsamer als FFs

Klassifizierung I/III

- Eine Klassifizierungsmöglichkeit von Halbleiterspeichern ist die Art des Zugriffs
  - Speicher mit wahlfreiem Zugriff (Matrixspeicher, random access memory)
    - Die einzelnen Zellen werden matrixartig angeordnet
  - Speicher mit seriellem Zugriff
    - Das Ein- und Auslesen erfolgt seriell (keine individuelle Adressierung)
    - FIFO (First-in-First-Out): Zuerst Eingelesene Daten werden zuerst wieder ausgelesen (schieberegisterartig)
      - Anwendung: Kommunikation von Funktionsblöcken mit unterschiedlicher Taktfrequenz

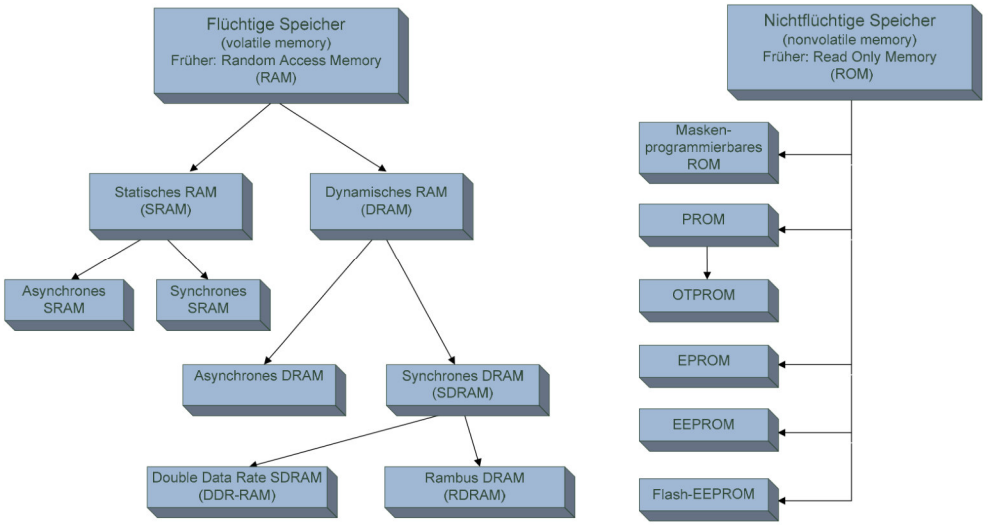
Klassifizierung II/III

- LIFO (Last-in-First-Out): Zuletzt Eingelesene Daten werden zuerst wieder ausgelesen
  - Anwendung: Stack in Mikroprozessor
- Assoziativspeicher (Inhaltsadressierbare Speicher, content addressible memory (CAM): Adressierung der Daten erfolgt durch spezielle Suchbegriffe
  - Anwendung: MAC zu Port in Netzwerkroutern, Cache in Mikroprozessorsystemen

- In der Praxis werden LIFOs, FIFOs und CAMs aber meistes durch Matrixspeicher realisiert

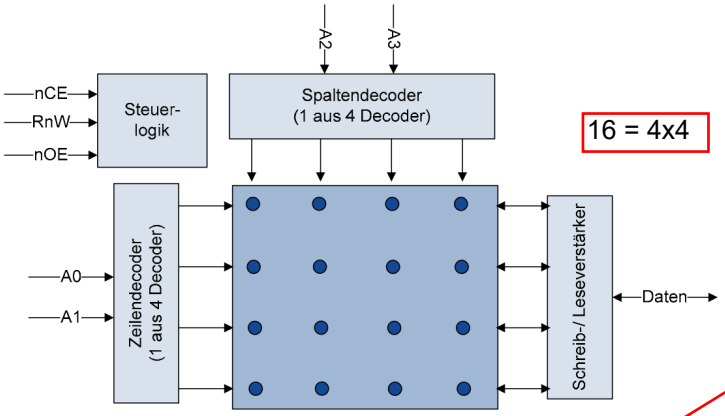
Klassifizierung III/III

- Eine weitere Klassifizierungsmöglichkeit von Halbleiterspeicher ist die Kenngroße der Datenverfügbarkeit (data rentention time)
  - Flüchtige Speicher (volatile memory, früher als RAM bezeichnet) verlieren Ihre Daten beim Ausschalten
  - Nichtflüchtige Speicher (non-volatile memory, früher als ROM bezeichnet) behalten Ihre Daten beim Ausschalten



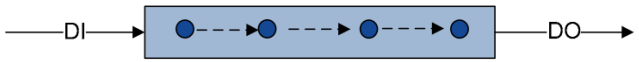
Speicherstrukturen - Matrixspeicher

- Beispiel Matrixspeicher 16x1
  - Die Abbildung zeigt einen Matrixspeicher mit 16bit, der mit vier Adressleitung angesteuert wird und am Datenausgang die Speicherzelle von einem Bit ausgibt (Wortbreite 1)
  - Die Adressleitungen werden bei einer quadratischen Matrix je zur Hälfte einem Spaltendecoder und einem Zeilendecoder verbunden
  - Für jede Adresse wird genau eine Zelle in der Matrix aktiviert
- Beispiel Matrixspeicher 16x8 (Wortbreite 8)
  - Bei einem Matrixspeicher 16x8 beträgt die Größe der Speicherzelle 8bit -> 16 Zeilen a 8bit und Ausgangsgröße 8bit



- nCE (not chip enable): nCE = 0 -> chip aktiv
- RnW (read not write): RnW = 1 -> lesen; RnW = 0 -> schreiben.
- nOE (not output enable): nOE = 0 -> Ausgänge aktiv (Three-State-Treiber an).

Speicherstrukturen - serieller Speicher (B)



- Beispiel serieller FIFO (schieberegisterartig ohne Adresseingänge)
  - In dieser Struktur ohne Adresseingänge kann nicht auf die einzelnen Speicherelemente zugegriffen werden
  - Eingangsdaten werden immer in der erste Speicherzeller abgelegt und die letzte Speicherstelle wird am Ausgang ausgegeben

Kennzahlen I/II

- Speicherkapazität und Struktur des Matrixspeichers NxM
  - Wortbreite M= Breite des Datenbusses in Bit
  - Anzahl Speicherworte N
  - Die gesamte Speicherkapazität ergibt sich durch durch N*M und wird angegeben in Bit (Abkürzung b) bzw. Byte (Abkürzung B) ggf. mit Präfix
    - k(klio): 2^10=1024
    - M(Mega) 2^20=1 048 576
    - G(Giga) 2^30=1.073742* 10^9
    - T(Tera) 2^40= 1.099511627776*10^12

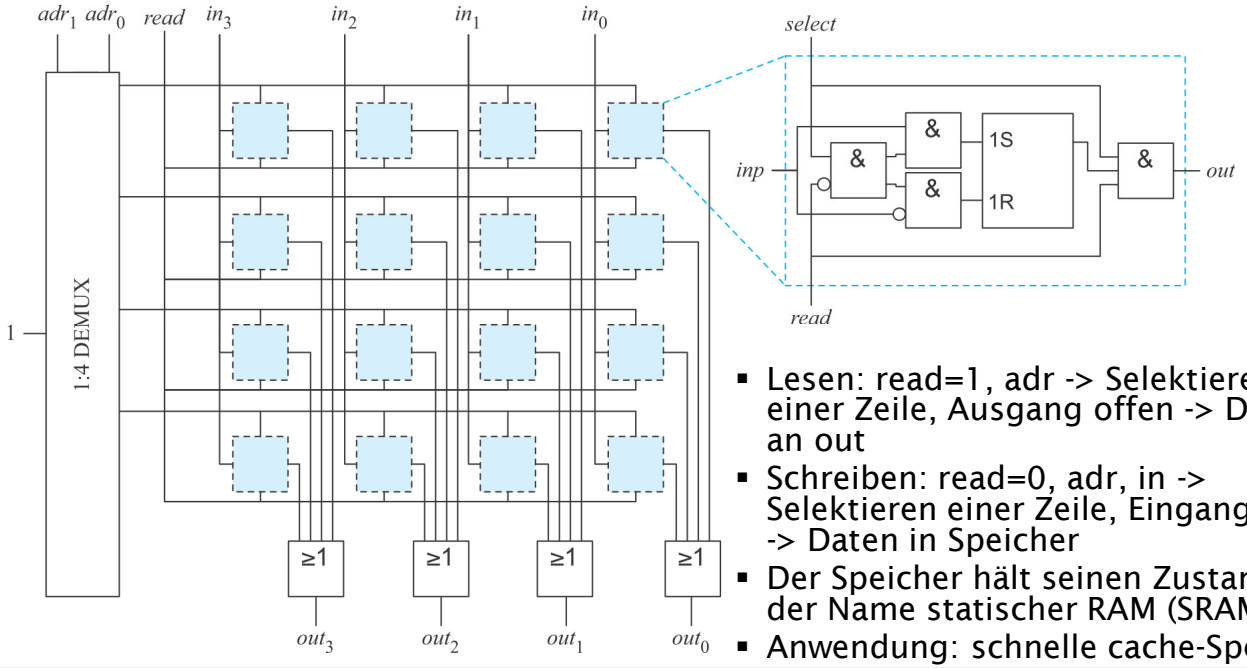
meistens klein b

Kennzahlen II/II

- Elektrische Verlustleistung (mw/Zelle)
- Speicherzugriffzeit (access time)= Zeit zwischen Ansteuerung(Adressierung) und Ende des Datentransfers
- Speicherzykluszeit= Zeitraum zwischen zwei aufeinander folgender Lesezugriffe oder Schreibzugriffe
- Data rentention= Zeit des Datenerhalts einer Speicherzelle
- Endurance= Menge an fehlerfreien Speicher bzw. Löschzyklen (Lebensdauer)

Flüchtige Speicher - SRAM - Prinzip

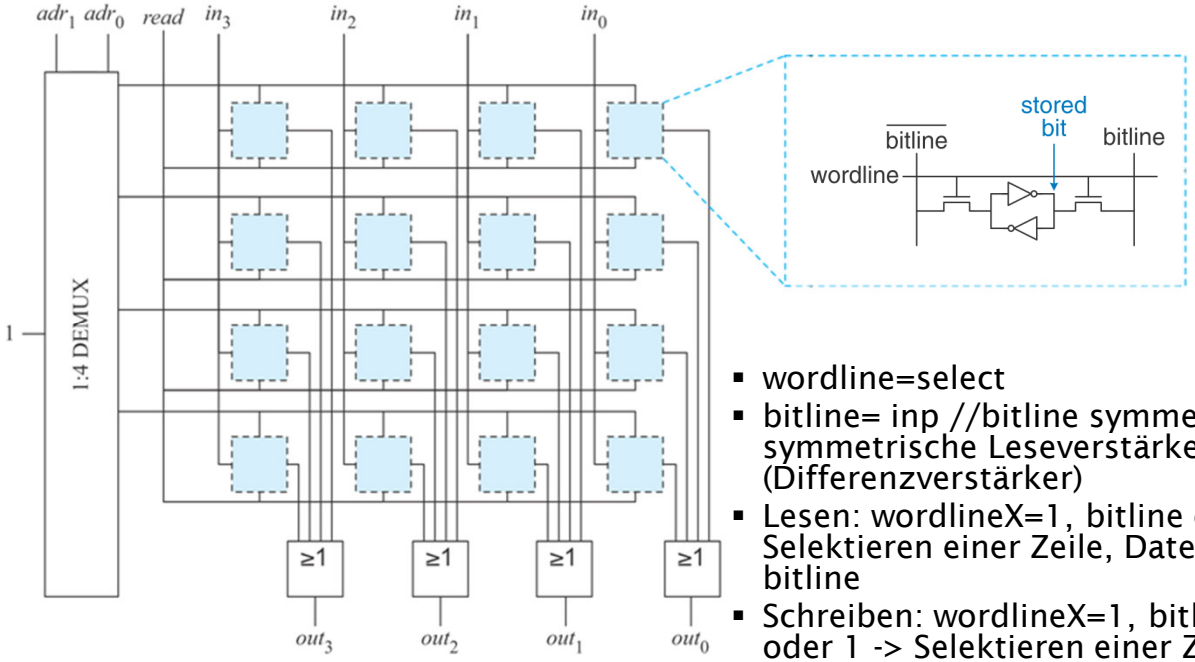
- Beispiel 4x4 SRAM (Static Random Access Memory)



- Lesen: read=1, adr -> Selektieren einer Zeile, Ausgang offen -> Daten an out
- Schreiben: read=0, adr, in -> Selektieren einer Zeile, Eingang offen -> Daten in Speicher
- Der Speicher hält seinen Zustand -> der Name statischer RAM (SRAM)
- Anwendung: schnelle cache-Speicher

Flüchtige Speicher - SRAM - Prinzip - Optimierte Umsetzung einer SRAM Zelle

- Beispiel 4x4 SRAM (Static Random Access Memory) - Optimierte Umsetzung einer SRAM Zelle (4 Trs für inverters + 2 Trs=6 Trs)

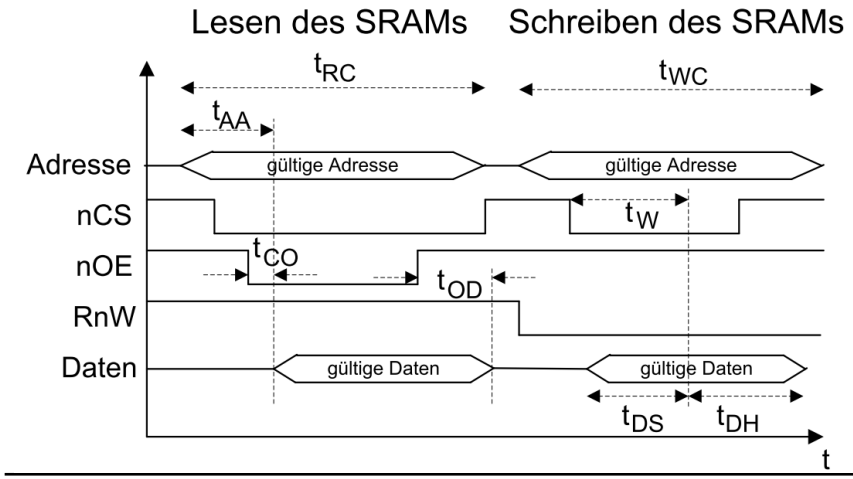


- wordline=select
- bitline= inp //bitline symmetrisch für symmetrische Leseverstärker (Differenzverstärker)
- Lesen: wordlineX=1, bitline offen -> Selektieren einer Zeile, Daten an bitline
- Schreiben: wordlineX=1, bitline=0 oder 1 -> Selektieren einer Zeile, bitline in Speicher

1 drittel des Platzbedarfs eines Registers

Flüchtige Speicher - SRAM - Timing und Ansteuerung - Lesen

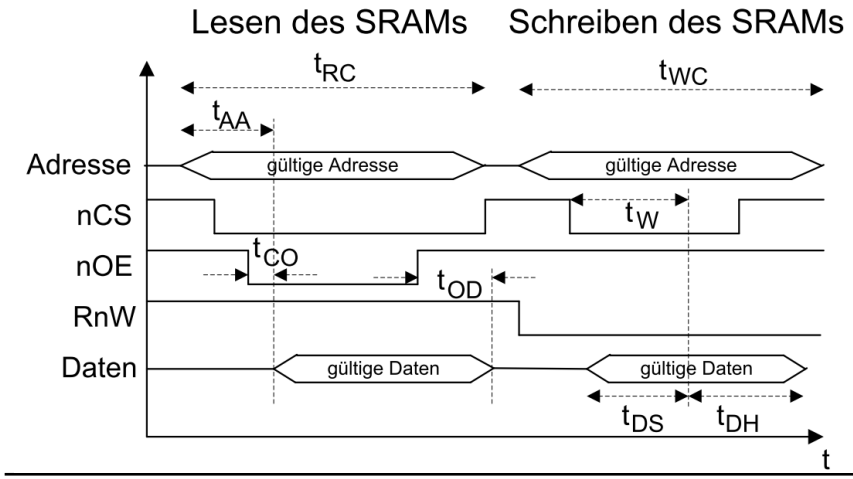
- nCS (not chip enable): !CE = 0 -> chip aktiv
- nOE (not output enable): !OE = 0 -> Ausgänge aktiv (Tri-State-Treiber an)
- RnW (read not write): RnW = 1 -> lesen; RnW = 0 -> schreiben



- Erklärung Zeiten Lesen
  - Trc read cycle time/ Lese-Zyklus-Zeit
  - Taa address access time / Adress-zugriffszeit (gültige Adresse bis Daten auf Ausgang)
  - Tco (Zeit die von nOE=1 bis Daten auf Bus liegen)
  - Tod (Zeit die Daten noch auf dem Bus liegen nachdem nOE=1)
- Lesen: Adr stabil, RnW=1, nCS=0 -> nOE=0 -> Warten Taa, Tco -> Daten lesen

Flüchtige Speicher - SRAM - Timing und Ansteuerung - Schreiben

- nCS (not chip enable): !CE = 0 -> chip aktiv
- nOE (not output enable): !OE = 0 -> Ausgänge aktiv (Tri-State-Treiber an)
- RnW (read not write): RnW = 1 -> lesen; RnW = 0 -> schreiben

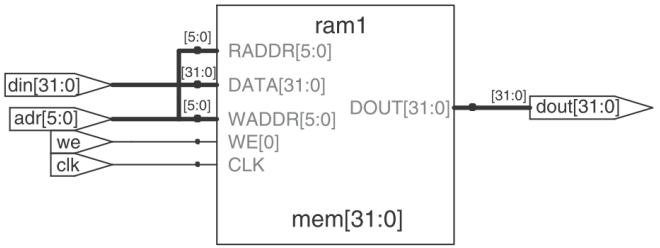


- Erklärung Zeiten Schreiben
  - Twc write cycle time / Schreibzykluszeit
  - Tds und Tdh (setup und hold time für Daten, Zeiten an denen Daten stabil sein müssen)
  - Tw (Breite des !CS Pulses)
- Schreiben: Adr stabil, RnW=0, nCS=0 -> Daten stabil -> Warten Tw, Tds -> Warten Tdh

Flüchtige Speicher - SRAM - VHDL

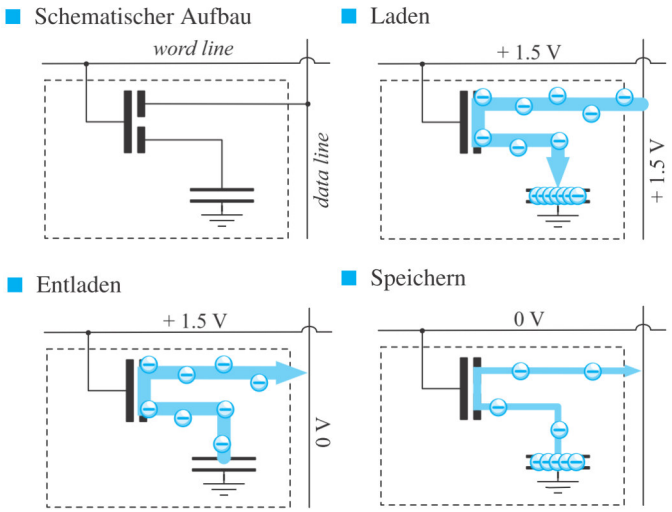
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD_UNSIGNED.ALL;

entity ram_array is --NxM=64x32 --
NxM=2^6x32
  port(
    clk, we: in STD_LOGIC;
    adr: in STD_LOGIC_VECTOR(5 downto 0);
    din: in STD_LOGIC_VECTOR(31 downto 0);
    dout: out STD_LOGIC_VECTOR(31 downto 0)
    );
   end;
architecture arch of ram_array is
  type mem_array is array ((2**6 - 1)
downto 0)
       of STD_LOGIC_VECTOR (31 downto 0);
  signal mem: mem_array;
begin
```

```
process(clk) begin
  if rising_edge(clk) then
    if we='1' then
      mem(TO_INTEGER(adr)) <= din;
    end if;
  end if;
end process;
dout <= mem(TO_INTEGER(adr));
end;
```



Flüchtige Speicher - DRAM - DRAM-Zelle - Prinzip

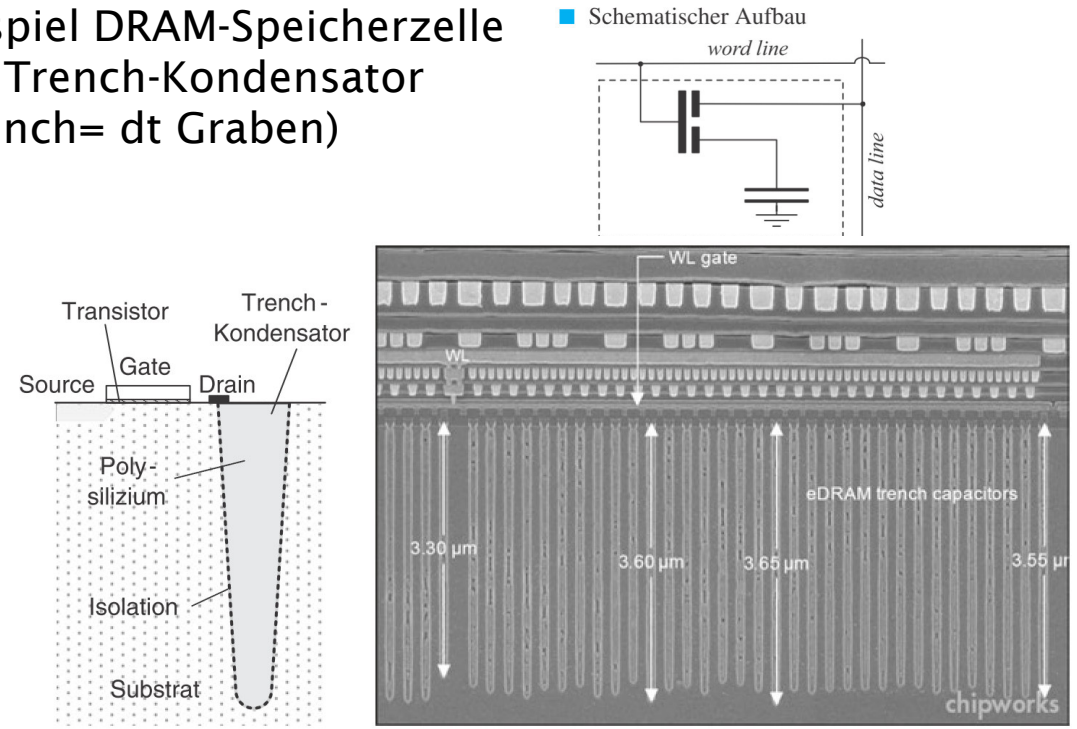
- DRAM-Speicher (Dynamic Random Access Memory) speichert ein einziges Bit durch eine kleine Ladung, die durch einen einzigen Transistor gesteuert auf einen Kondensator geladen/entladen wird
- Laden (1 Schreiben): word line=1, data line=1 -> Trs schaltet -> C wird geladen
- Entladen (0 Schreiben): word line=1, data line=0 -> Trs schaltet -> C wird entladen
- Speichern: word line=0 -> Trs sperrt
  - Da Trs nicht ideal sperrt (Leckströme) entläd sich die Ladung langsam -> wenn Zustand der Zelle gespeichert werden soll muss Inhalt zyklisch ausgelesen und neu beschrieben werden (DRAM refresh)
- Lesen: Precharge bit line (definierte Spannung) -> bit line hochohmig -> word line=1 -> Ladung ändert Spannung auf bit line
  - Nach Lesen muss ebenfalls ein refresh gemacht werden



- Anwendung: größere Speicher wie z.B. PC Hauptspeicher (höhere Speicherdichte (nur 1 Trs) aber langsamer als SRAM (auf Umladung warten, refresh)

Flüchtige Speicher - DRAM-Zelle - physikalische Realisierung (B)

- Beispiel DRAM-Speicherzelle mit Trench-Kondensator (Trench= dt Graben)

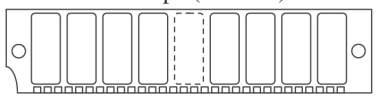


Flüchtige Speicher - DRAM - Module

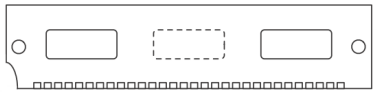
- Computersysteme verwenden DRAM in Form von standardisierten Speichermodulen
- SIMM (Single Inline Memory Module) haben Speicherchips auf der Vorder- und Rückseite, wobei die Kontakte der Vorder- und Rückseite verbunden sind
- DIMM (Dual Inline Memory Module) haben Speicherchips auf der Vorder- und Rückseite, wobei die Kontakte der Vorder- und Rückseite nicht verbunden sind
- Typische Speicherchips besitzen 1,2,4,8 oder 16 Datenausgänge (x1, x2, x4, x8, x16 Speicher)
- Generell ist die Anzahl der Datenausgänge pro Chip multipliziert mit der Anzahl der Speicherbausteine auf dem Modul die Breite des Datenbusses
- Beispiel:
  - SIMM 30Pin: 8 Chips x 1bit=8bit; 2 Chips x 4bit= 8bit
  - DIMM 240pin (DDR2/DDR3): 8 Chips x 8bit=64bit; 4 Chips x 16bit=64bit
- Rank: Bei single rank wird die Busbreite (z.B.64bit) von den Speicherchips nur 1x abgedeckt; Bei dual Rank wird die Busbreite zweimal abgedeckt -> mit billigeren Chips gleiche Speichermenge, aber nicht so störsicher

- SIMM-Modul (30 Pins, 8 Bit)

×1-Chips (8 Stück):

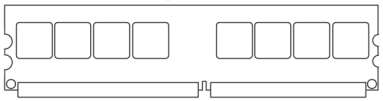


×4-Chips (2 Stück):

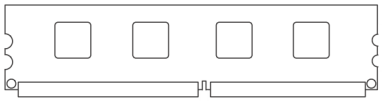


- DIMM-Modul (240 Pins, 64 Bit)

×8-Chips (8 Stück):

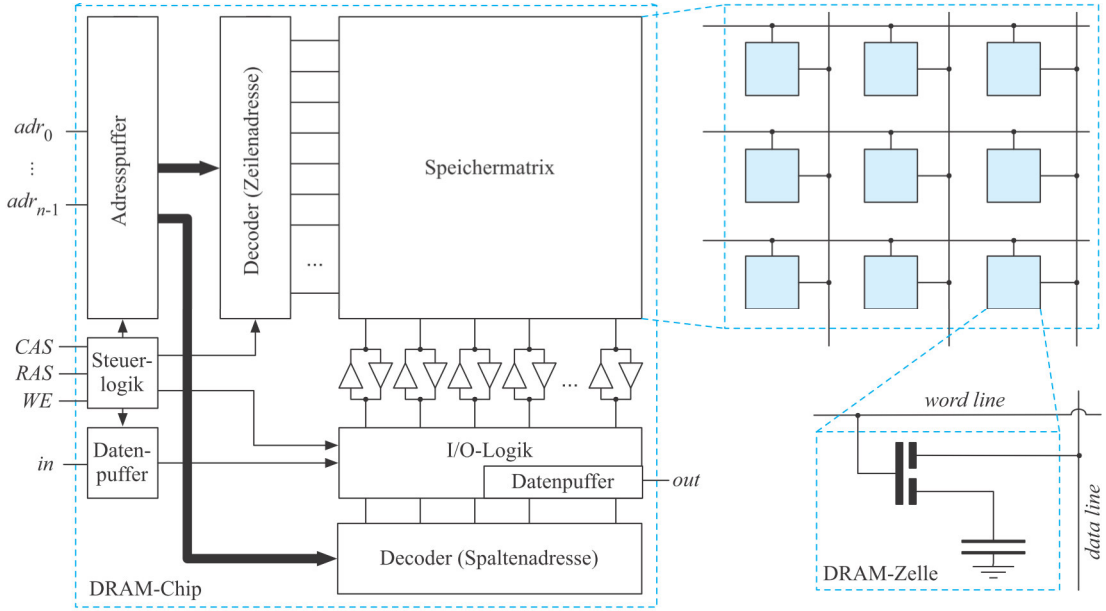


×16-Chips (4 Stück):



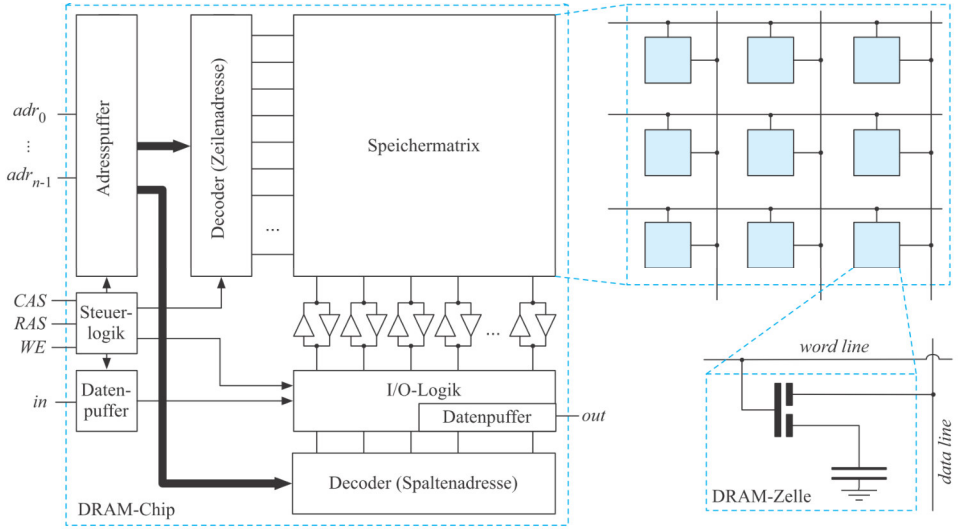
single ranked immer besser
for allem bei Signalqualität

Flüchtige Speicher - DRAM - DRAM-Chip Aufbau I/II



- Die Speicherzellen sind in einer Speichermatrix organisiert
- Alle Zellen einer Zeile nennt man page
- Die kleinste adressierbare Speichereinheit (KAE) ist ein Bit, oft aber auch vielfache davon (z.B. 8,16,32 Bit)

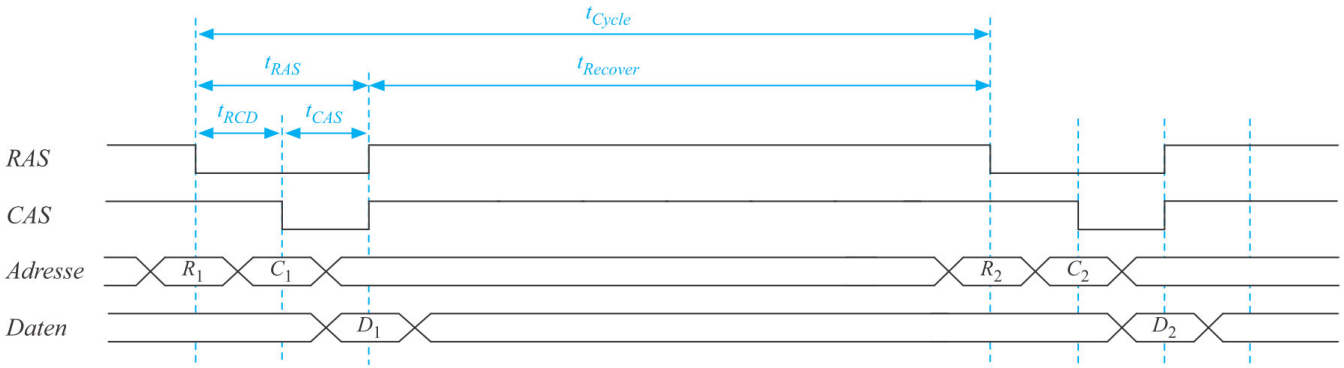
Flüchtige Speicher - DRAM - DRAM-Chip Aufbau II/II



- Die Adresse der KAE setzt sich aus einem Zeilenteil und einem Spaltenteil zusammen
- in der Regel wird die Zeilenadresse und Spaltenadresse nacheinander über die gleichen Adressleitung eingelesen (address multiplexing)
- Die Steuersignale RAS (Row Address Strobe) und CAS (Column Address Strobe) legen fest, wie die angelegte Adresse interpretiert wird (historisch low active - Beispiel Zeilenadresse -> CAS=0)

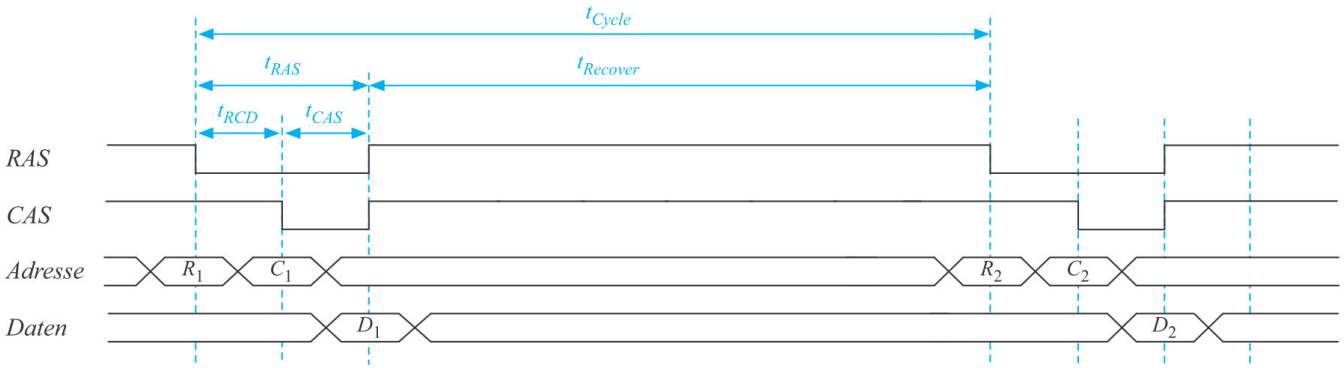
Steuersignale sind low aktiv

Flüchtige Speicher - DRAM -Timing und Ansteuerung I/II



- Erklärung Zeiten: Trcd row column delay (Wartezeit zwischen Anlegen row address und column address); Tcas (Wartezeit nach CAS bis Daten gültig); Tras (Zugriffszeit, Trcd+Tcas); Trecover (Wartezeit nach Zugriff bis nächster Zugriff); Tcycle (minimale Zeit zwischen 2 Zugriffen; Tras+Trecover)

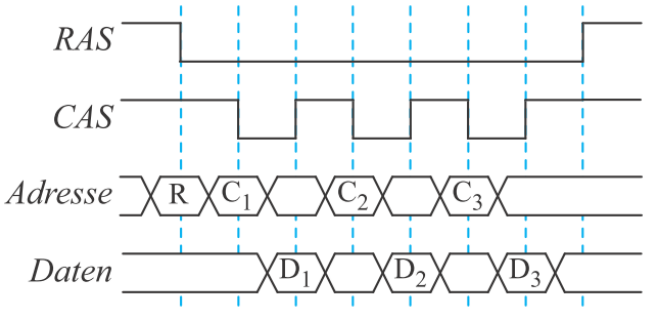
Flüchtige Speicher - DRAM -Timing und Ansteuerung II/II



- DRAM-Read: !WE=1, row address stabil, RAS=0 -> Warten von Trcd -> column address stabil, CAS=0 -> Warten Tcas (insgesamt Tras) -> Daten lesen-> warten Trecover
- DRAM-Write: !WE=0, row address stabil, RAS=0 -> Warten von Trcd -> column address stabil, CAS=0 -> Warten Tcas (insgesamt Tras) -> Daten stabil-> warten Trecover
- In beiden Fällen muss die komplette page zurückgeschrieben werden (refresh)

Flüchtige Speicher - DRAM - Optimierung - (fast) page mode

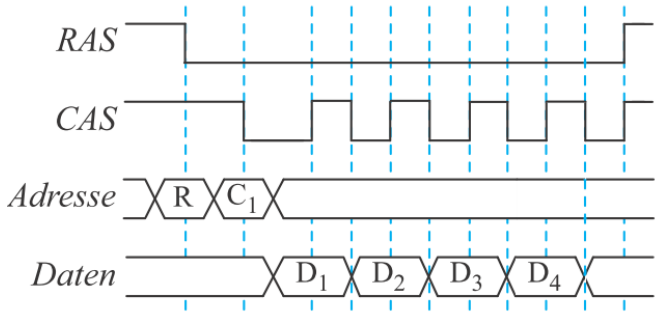
- Da Tcycle recht lange ist, wurden einige Optimierungen eingeführt
- Idee (fast) page mode: Zwei aufeinander folgende Zugriffe beziehen sich oft aufeinander folgende Adressen (z.B. array-Zugriff in for-Schleife) -> Beibehalten von Zeilenadresse



- (fast) page mode:
  - RAS-Signal bleibt über mehrere Zugriffe hinweg aktiviert (Zeilenadresse bleibt die gleiche es wird nur noch die Spaltenadresse übertragen)
  - Ab dem zweiten Zugriff verringert sich die Zugriffszeit von Tras auf Tcas

Flüchtige Speicher - DRAM - Optimierung - burst mode (nibble mode)

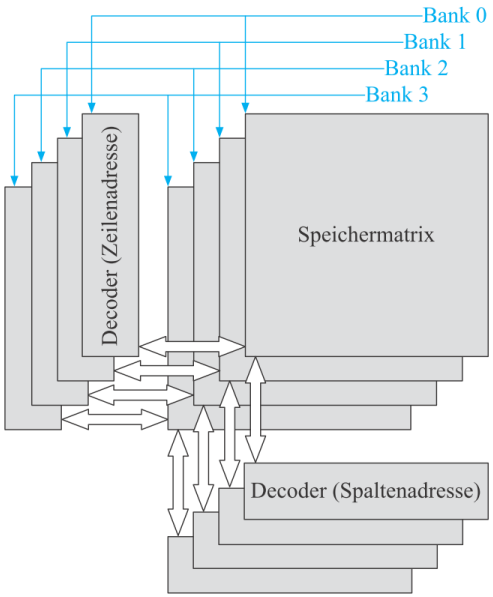
- Idee burst mode (nibble mode):
  Bei aufeinander folgenden Zugriffen Beibehalten von Zeilenadresse und Spaltenadresse eines integrierten Zählers



zugriff Beibehalten von Zeilenadresse und Spaltenadresse (intern wird die Spaltenadresse mit Hilfe eines Zählers inkrementiert)

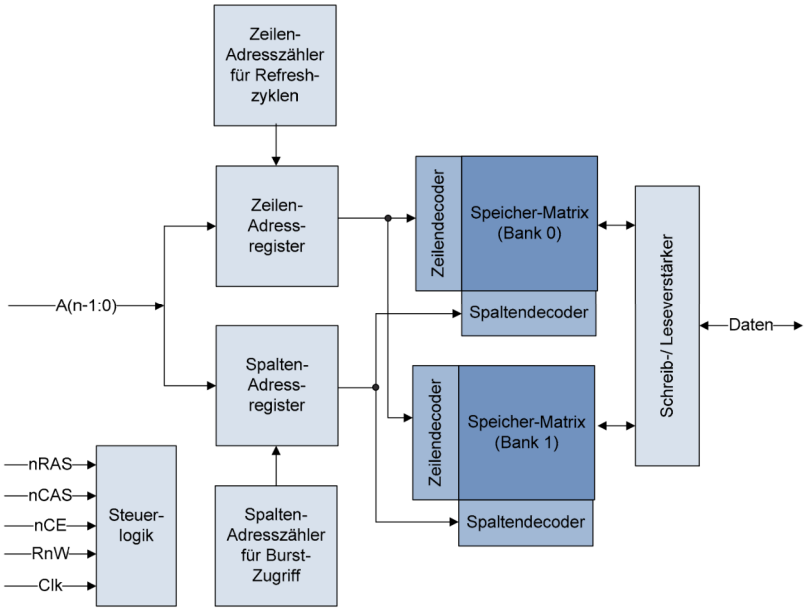
Flüchtige Speicher - DRAM - Optimierung - Interleaving / bank switching

- Das größte Performanceproblem der Langen Tcycle stellt immer noch die lange Trecovery, die Zugriffe hinter einander limitieren
- Idee interleaving: Aufteilen des Speicherchips in eine Zweierpotenz von Speicherbänken mit eigenen RAS und CAS Leitungen, Speichermatrix und I/O-Logik
- Jede Bank ist quasi ein eigener Chip
- Speicherzellen werden so auf die verschiedenen Segmente verteilt, dass ein Zugriff auf hintereinander folgender Speicheradressen die Bänke in abwechselnder Reihenfolge angesprochen werden (interleaving/ bank switching)
  - Während sich eine Bank erholt, können andere Bänke benutzt werden
- Vorteil: Geschwindigkeitsgewinn ist enorm (fast nur noch durch Tras begrenzt)
- Nachteil: höhere Komplexität und Kosten durch mehrfache Ansteuerlogik



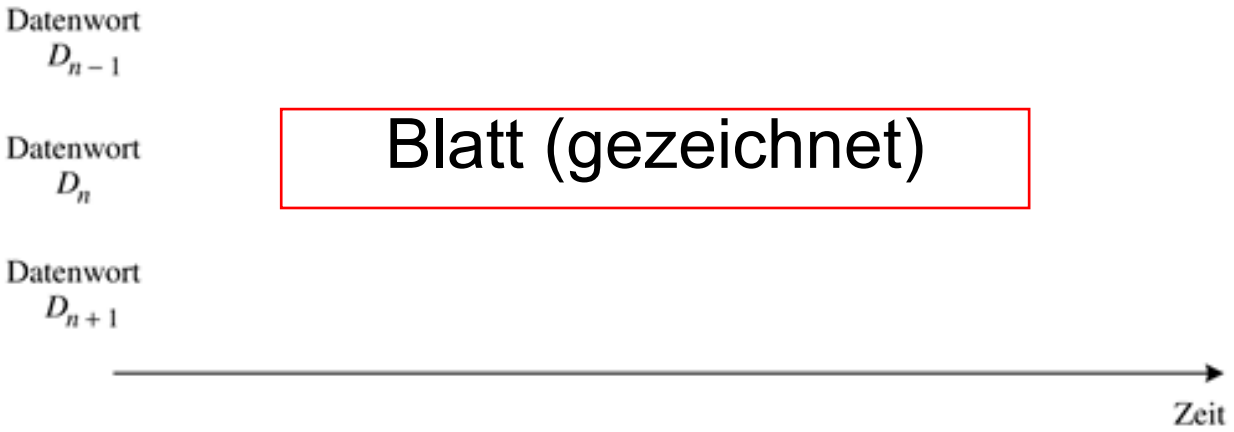
Flüchtige Speicher - DRAM - Optimierung - SDRAM

- Die bisher betrachteten DRAM-Varianten sind asynchron und nutzen alle kein CLK-Signal
- SDRAM (synchronous DRAM) kann deutlich höhere höhere Geschwindigkeit erzielt werden
  - Pipelining von Zugriffen
  - Vereinfachung des das Handling mit RAS und CAS (Änderungen nur noch bei positiven Taktflanken)
- Weiterhin verfügen SDRAMs über 2 interne Speicherbänke



Flüchtige Speicher - DRAM - Optimierung - SDRAM - Übung

- Skizzierten Sie das Pipelining von Zugriffen beim SDRAM
- Zerlegen Sie den DRAM Zugriff in die zwei Teilschritte Adressieren und Lesen/Schreiben

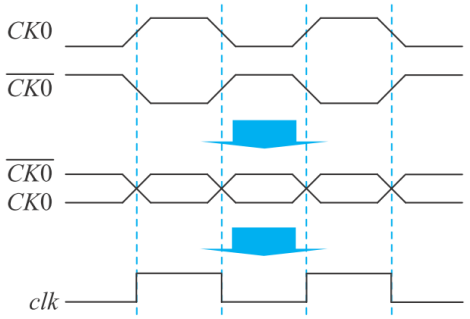


Teil 2

Flüchtige Speicher - DRAM - Optimierung - DDR-RAM

- Die Abkürzung SDRAM wird manchmal auch als Single Data Rate RAM interpretiert
- Dieser Speicher arbeitet „einflankengesteuert“
- DDR-RAM (Double Data Rate RAM) verwendet neben der positiven Flanke auch die negative Flanke zur Datenübertragung
  - Es wird intern immer das Doppelte der Datenmenge aus dem Speicher gelesen und gebuffert (prefetching), die (von einem SDRAM) bei der steigenden Flanke ausgegeben werden kann
  - Bei der fallenden Flanke wird der Rest ausgegeben

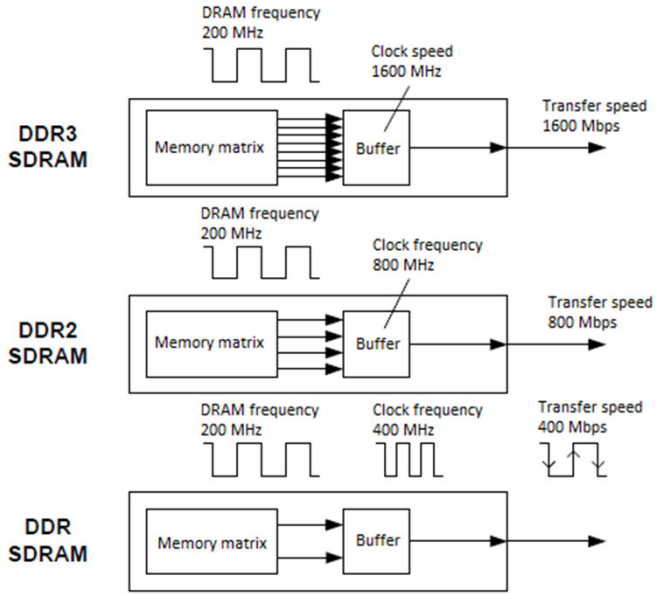
- Das Taktsignal wird differentiell übertragen, was zwar 2 Pins statt 1 Pin kostet aber deutlich genauer und weniger Störanfällig ist



- Aus dem gleichen Grund wird das DQS-Signal (data strobe; eine Art data valid Signal) differenziell übertragen

Flüchtige Speicher - DRAM - Optimierung - DDR-RAM - Evolution DDRx

- DDR: 100-200MHz Speichertakt, 100-200 MHz I/O-Takt, 2-fach prefetch, 2.5/2.6V, DIMM 184 Pins
- DDR2: 100-266 MHz Speichertakt, 200-533 MHz I/O-Takt, 4-fach prefetch, 1.8V, DIMM 240pin
- DDR3: 100-266 MHz Speichertakt, 400-1066 MHz I/O-Takt, 8-fach prefetch, 1.5/1.35V, DIMM 240pins
- DDR4: 200–400 MHz Speichertakt, 800-1600 MHz I/O-Takt, 8-fach prefetch, 1.05/1.2V, DIMM 288pins
- DDR5: 200–525MHz Speichertakt, 1600-2400 MHz I/O-Takt, 16-fach prefetch (32-fach prefetch optional), 1.1V, DIMM X Pins
- Modulbezeichnung PCx-xxx oder Speichertransferrate (in MByte/s)= (Speichertakt (in MHz) × Busbreite (in Bit) × Prefetching-Faktor) / (8 Bit/Byte)

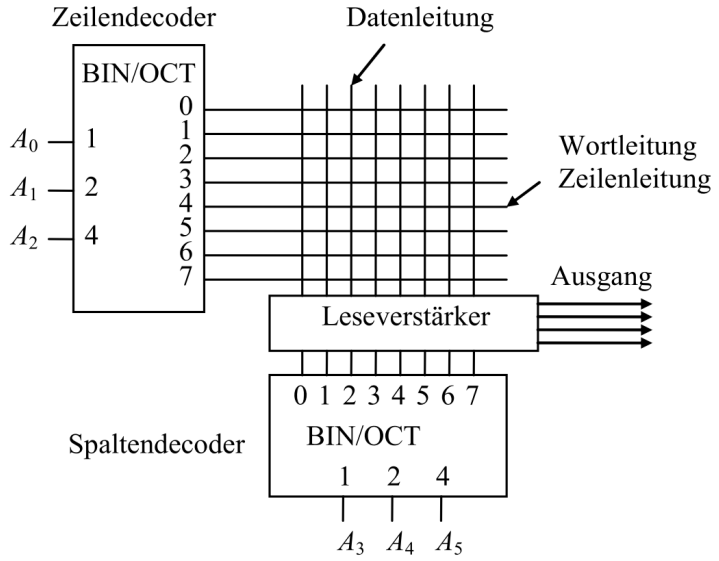


Doppelt soviel Buffern mit doppelt so schnellem Buffer

Nicht flüchtige Speicher - ROM - Prinzip

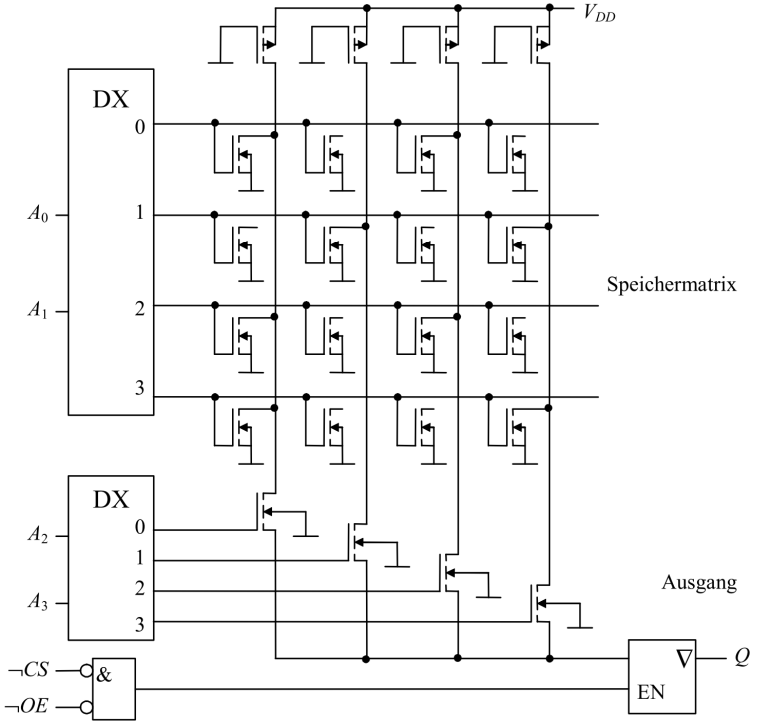
- ROM (Read Only Memory) ist ein Speicher dessen Dateninhalt schon vom Hersteller durch Masken definiert ist
- Die Daten sind daher fest und nicht flüchtig
- Die Daten können aber nur gelesen werden
- ROM-Speicher gibt es mit eine Wortlänge von 1,4,8,16 bit
- Auch das Rom ist matrixartig organisiert (vgl. Beispiel 64x4 ROM)

- Die Speicherzellen liegen an den Schnittpunkten der Leitungen
- Die Speicherzellen werden angesprochen, wenn die Zeilenleitung (word line) und die Spaltenleitung (bit line oder data line) auf 1 liegen



Nicht flüchtige Speicher - ROM - Realisierung

- Beispiel 16x1 ROM in CMOS
  - Das Rom besteht aus 16 n-Kanal MOSFETs (pro Speicherzelle 1 Trs)
  - Beispiel Lesen Adresse 1101
    - Auswahl word line 1 und bit line 3 -> Trs schaltet -> 0
  - Beispiel Lesen Adresse 0110
    - Auswahl word line 2 und bit line -> Trs nicht angeschlossen, nichts schaltet -> 1



Nicht flüchtige Speicher - ROM - VHDL

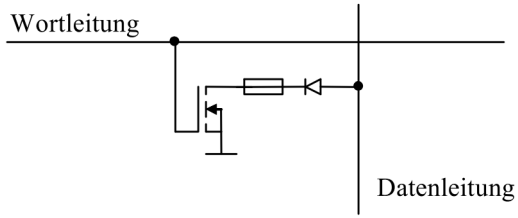
```
library IEEE; use
IEEE.STD_LOGIC_1164.all;
entity rom_case is
  port(adr: in STD_LOGIC_VECTOR(1
                              downto 0);
       dout: out STD_LOGIC_VECTOR(2
                             downto 0));
end;
architecture arch of rom_case is
begin
process(adr) begin
  case adr is
    when "00" => dout <= "011";
    when "01" => dout <= "110";
    when "10" => dout <= "100";
    when "11" => dout <= "010";
  end case;
end process;
end;
```

```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ROM_ARRAY is
   port(ADDRESS : in std_logic_vector(3
                                    downto 0);
     DATA : out std_logic_vector(7
                                    downto 0));
end;

architecture ARCH of ROM_ARRAY is
  type ROM_TYPE is array (0 to 2**3) of
    std_logic_vector(7 downto 0);
  constant ROM : ROM_TYPE := (x"0F",x"0E",
      x"0D",x"0C",x"0B",x"0A",x"09",x"08",
      x"07",x"06",x"05",x"04",x"03",x"02",
      x"01",x"00");
begin
   DATA <=
        ROM(to_integer(unsigned(ADDRESS)));
end;
```

Nicht flüchtige Speicher - PROM - Prinzip

- Ein PROM (Programmable ROM) ist ein programmierbarer ROM
- Der Aufbau eines PROMs ist genauso wie ein ROM matrixartig mit Zeilen und Spaltendekoder
- Die Drains der Transistoren in den Speicherzellen werden anstatt einer Leitung mit einem „fusible link“ (einer Art Schmelzsicherung) kontaktiert

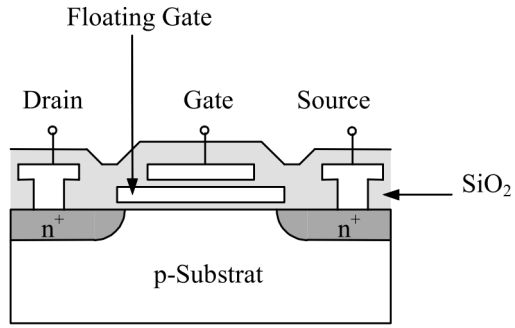
- Das Programmieren der Zelle entspricht dem Schmelzen der Sicherung durch Anlegen von höherer Spannung
- Im nicht-programmierten Zustand gibt die Zelle 0 zurück
- Wenn die Zelle programmiert ist, wird 1 zurückgegeben

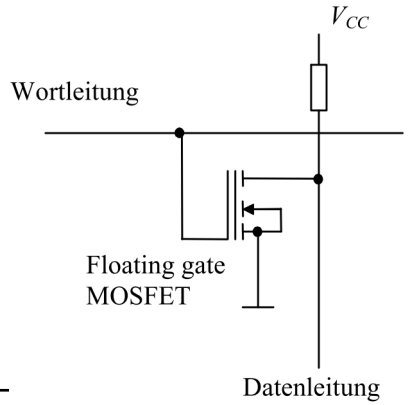


- Die Zelle ist nur einmal programmierbar (Sicherung ist geschmolzen!), daher wird das PROM auch als OTP-ROM (one time programmable ROM) genannt

Nicht flüchtige Speicher - EPROM - Prinzip I/II

- Das EPROM (eraseable PROM) hat einen ähnlichen Aufbau wie das PROM
- Der Unterschied ist, dass an Stelle der „fuseable links“ löschbare Speicherelemente liegen
- EPROMs verwenden Floating-Gate-MOSFETs (auch FAMOS (Floating Gate Avalanche MOS) Transistor genannt)
- Floating-Gate-MOSFETs sind NMOS Transistoren mit einem zusätzlichen Gate, das keine Verbindung nach draußen hat (floating gate)
- Durch eine Ladung auf dem floating gate kann Information gespeichert werden
- Ohne Ladung funktioniert der Transistor wie ein normaler NMOS-Transistor (positive Spannung am Gate -> Schalten -> 0)
- Mit einer negativen Ladung auf dem floating gate sperrt der Transistor trotz anlegen einer positiven Spannung am Gate (Sperren -> 1)





Nicht flüchtige Speicher - EPROM - Prinzip II/II

- Programmiert wird das EPROM durch Tunnelung von Elektronen durch das Oxid
  - Das wird erreicht durch eine erhöhte Spannung zwischen Drain und Substrat
  - Das elektrische Feld (Spannung) zwischen Gate und Kanal erreicht dabei so hohe Werte das es zum Avalanche-Effekt (Lawinen-Effekt) kommt
  - Eine gewisse Anzahl von Elektronen kann dabei durch das Gate-Oxid auf die Floating-Gate-Elektrode tunneln
  - Dadurch entsteht eine negative Ladung auf dem Gate, die den Transistor sperrt
- Gelöscht wird das EPROM durch eine etwa 20-minütige Bestrahlung mit UV-Licht
  - Die UV-Bestrahlung ionisiert das Gate-Oxid, wodurch die Ladung wieder abfließen kann
  - Dadurch ist die Information wieder gelöscht
- Die Ladungsspeicherung selbst ist durch die guten Eigenschaften des Oxids auf Jahre stabil

Nicht flüchtige Speicher - EEPROM - Prinzip

- Um den Nachteil des komplizierte Vorgangs des Löschen des EPROMs durch UV-Licht zu vermeiden wurden EEPROMs entwickelt
- Das EEPROM (electronically eraseable PROM) ist elektronisch programmierbar und löschbar
- Ähnlich wie beim EPROM ist die Speicherzelle aus einem Floating-Gate MOSFET aufgebaut
- Die Dicke des Oxids zwischen Floating-Gate und Kanal ist allerdings wesentlich dünner
- Dadurch kann mit einer erhöhten Spannung Elektronen vom Kanal zum Gate oder vom Gate zum Kanal transportieren (Fowler-Nordheim Tunnel)

Nicht flüchtige Speicher - Flash (EEPROM) - Prinzip

- EEPROMs die nur insgesamt oder blockweise löschbar sind werden Flash-Speicher genannt
- Das erzwungene Löschen ganzer Blöcke auf einmal hat zu dem Namen „Flash“ geführt
- Ein Vorteil ist der geringere Schaltungsaufwand (nicht jede Zelle muss beim Löschen einzeln angesprochen werden können)
- Die Anzahl möglicher Löschzyklen ist begrenzt
- Bei der Ansteuerung wird versucht Blöcke möglichst gleich häufig zu verwenden, um die Lebensdauer zu erhöhen (wear leveling / Ausgleichen der Abnutzung)

Damit Lebenszeit von SSDs begrenzt

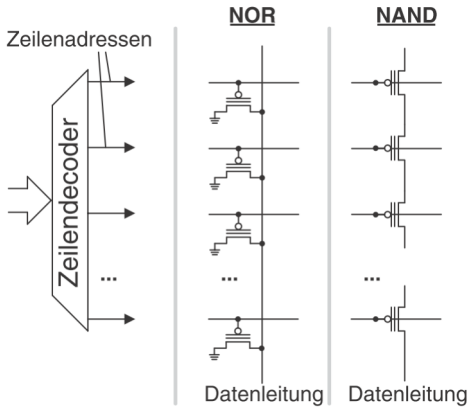
OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Nicht flüchtige Speicher - Flash (EEPROM) - Prinzip - NAND und NOR FLASH

- Es gibt zwei Strukturen für die Anordnung von Floating-Gate Transistoren (NOR und NAND-Struktur)
- Beiden Strukturen ist gemeinsam, dass wieder ein Zeilendekoder eine Zeile auswählt

- In der NOR-Struktur schalten alle Transistoren nach Masse
  - Die nichtaktiven Transistoren sind nicht leitend
  - Lesen: Ausgewählte Transistoren werden je nach Speicherzustand leitend oder nicht leitend
- In der NAND-Struktur sind die Speichertransistoren in der Datenleitung in Reihe geschalten
  - Die nichtaktiven Transistoren sind leitend (über mehr Spannung an den word Leitungen)
  - Lesen: Ausgewählte Transistoren werden je nach Speicherzustand leitend oder nicht leitend

urspr. Flashspeicher



Vorteil NOR:
durch schalten auf Masse - gute Lesbarkeit.
Nachteil:
Guter Kontakt zu Masse notwendig

Vorteil NAND:
keine Verbindung zu Masse -> weniger Platz
Nachteil:
Spannungsabfall durch Verkettung -> länge begrenzt da sonst nicht mehr Lesbar,
ein Glied kaputt -> ganze Reihe kaputt

Nicht flüchtige Speicher - Flash (EEPROM) - Prinzip - NAND und NOR FLASH - Vergleich

- Beide Strukturen werden praktisch eingesetzt
  - NOR:
    - Vorteil: gute Lesbarkeit der Daten (geringer Widerstand)
    - Nachteil: hoher Flächenbedarf (jeder Transistor braucht Kontakt zu Masse)
  - NAND:
    - Vorteil: geringer Flächenbedarf (direkt übereinander, kein Massekontakt)
    - Nachteil: schwierige Lesbarkeit der Daten (Trs ist auch im leitenden Zustand nicht ideal leitend -> Bei langen Ketten schwierig)
- Die meisten angebotenen Flash-Speicher verwenden aufgrund des Preisvorteils NAND-Technologie

NAND schlechter aber überall verwendet, da günstiger

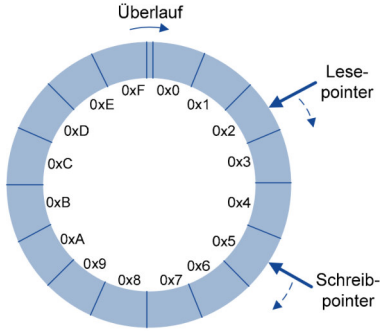
Nicht flüchtige Speicher - Flash (EEPROM) - Optimierung Speicherdichte

- Die Speicherdichte für Flash kann weiter verbessert werden, wenn man verschiedene Ladungsmengen auf das Floating-Gate speichert
- Je nach Ladungsmenge ergeben sich unterschiedliche Schwellspannungen die unterschieden werden können
- Diesen unterschiedlichen Ladungszuständen können dann unterschiedliche Bit-Zustande zugeordnet werden
- Somit können 2-4 bits pro Transistor gespeichert werden
- Bei 2bits spricht man oft von MLC-Flash (Multi-Level-Cell)
- Bei 3bits spricht man von TLC-Flash (Triple-Level-Cell)
- Bei 4bits spricht man von QLC (Quad-Level-Cell)
- Die klassische Technik mit 1bit nennt man im Vergleich dazu SLC (Single-Level-Cell)

Serielle Speicher - FIFO - Prinzip

- Das Prinzip ist ähnlich dem eines Schieberegisters
- Die am Eingang hineingeschobenen Daten müssen am Ausgang in der gleichen Reihenfolge wieder entnommen werden
- Allerdings sind keine feste Anzahl von Takten notwendig, sondern die lesende Anwendung definiert mit einem Freigabesignal selbst, wann die Daten ausgelesen werden sollen

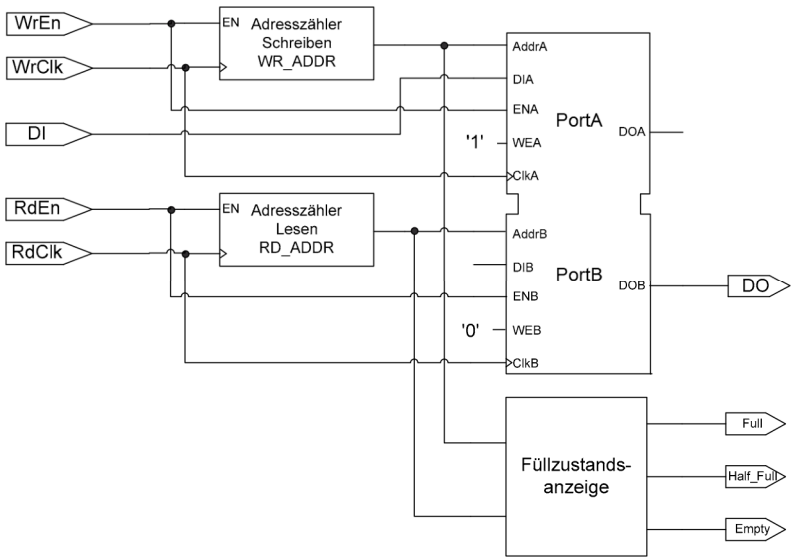
- Beispiel Arbeitsweise FIFO als Ringspeicher mit 16 Adressen



- Ein Schreibpointer adresisert die Speicherzelle, in die das nächste Datum zu schreiben ist
- Ein Lesepointer adressiert die Speicherzelle von der das nächste Datum ausgelesen wird

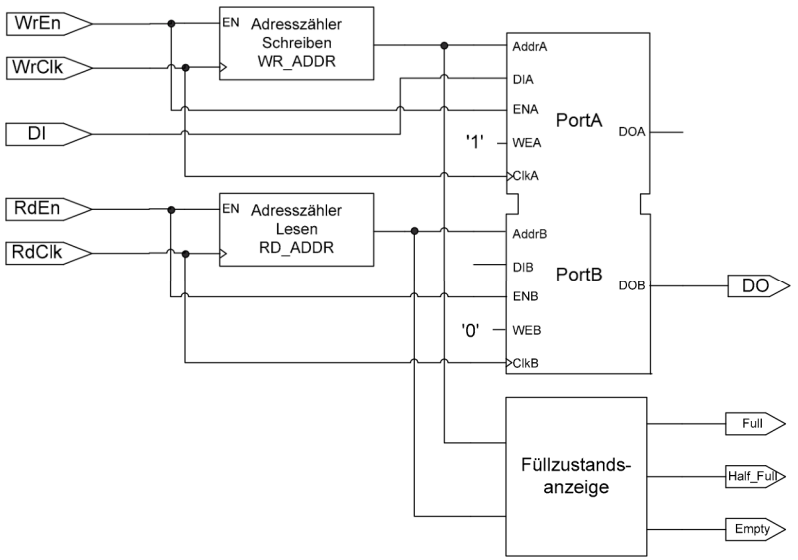
Serielle Speicher - FIFO - Umsetzung mit Dual Port SRAM I/II

- Schreiben des FIFOs über PortA:
  - Schreibpointer= zyklisch inkrementierender Adresszähler WR_ADDR (zeigt auf nächste freie Speicherstelle)
  - Sobald WrEn=1 -> pro Takt WR_ADDR=WR_ADDR+1
- Lesen ders FIFOs über PortB:
  - Lesepointer= zyklisch inkrementierender Adresszähler RD_ADDR (zeigt auf nächste auszulesende Speicherstelle)
  - Sobald RdEn=1 -> pro Takt RD_ADDR=RD_ADDR+1



Serielle Speicher - FIFO - Umsetzung mit Dual Port SRAM II/II

- Daneben gibt es eine Füllstandsanzeige, die dem schreibenden Prozess signalisiert, wenn der FIFO voll (full) ist und dem lesenden Prozess signalisiert, wenn der FIFO leer ist (empty)
  - empty= if (WR_ADDR == RD_ADDR)
  - full= if (WR_ADDR - RD_ADDR == Anzahl_Speicherelemente -1)
- Ein Überlauf bzw. Leerlauf kann nur dann vermieden werden, wenn die mittleren Datenraten beim Schreiben und Lesen gleich sind
- Die Füllstandssignale können dazu verwendet werden, die Datenraten der Quelle und Senke anzupassen (back pressure)



Serielle Speicher - FIFO - VHDL Dual-Port SRAM

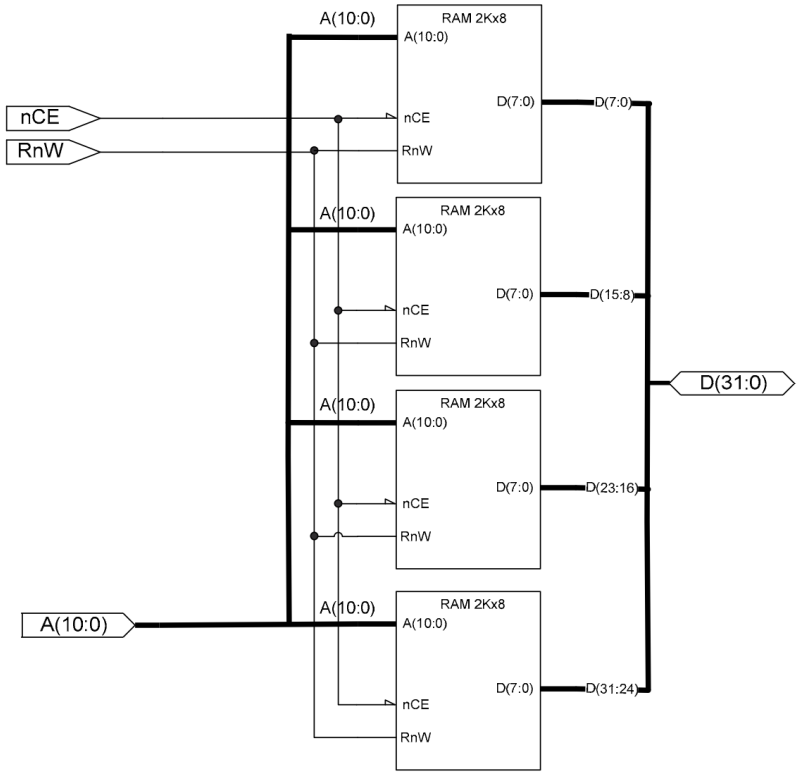
```
library ieee; use
ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity DPRAM is
port (CLKA : in bit;
  CLKB : in bit;
  WEA : in bit;
  ADDRA : in unsigned(5 downto 0);
  ADDRB : in unsigned(5 downto 0);
  DIA : in std_logic_vector(15 downto
0);
  DOB : out std_logic_vector(15 downto
0)
);
end;

architecture ARCH of DPRAM is
type RAM_TYPE is array (63 downto 0) of
  std_logic_vector (15 downto 0);
signal RAM : RAM_TYPE;
begin
PWRITE: process (CLKA)
begin
  if (CLKA'event and CLKA = '1') then
    if (WEA = '1') then
      RAM(to_integer(ADDRA)) <= DIA;
    end if;
  end if;
end process;
PREAD: process (CLKB)
begin
  if (CLKB'event and CLKB = '1') then
    DOB <= RAM(to_integer(ADDRB));
  end if;
end process;
end;
```

Speichersysteme - Motivation

- Insbesondere bei embedded systems werden mehrere Speicherbausteine eingesetzt
- z.B. EEPROM/Flash für Programm-Code, SRAM als schneller Zwischenspeicher und DRAM als Hauptspeicher
- Bei der Speicherkaskadierung unterscheidet man Wortbreitenerweiterung (Parallelerweiterung) und Speicherkapazitätserweiterung

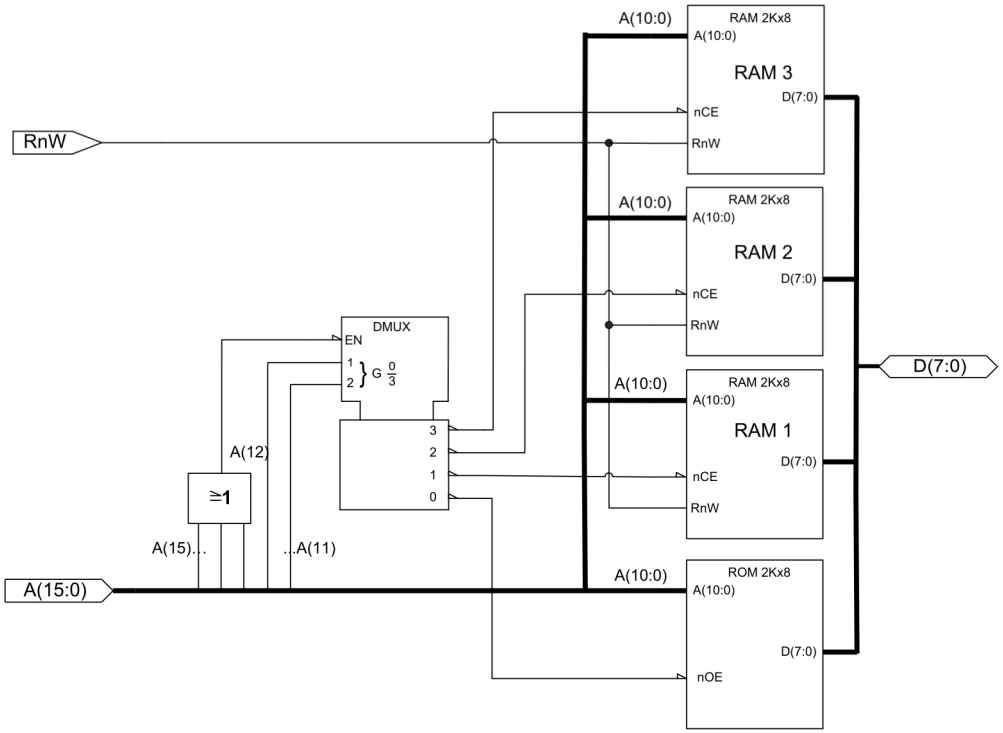
Speichersysteme - Speicherkaskadierung - Wortbreitenerweiterung (Parallelerweiterung)

- Unter Wortbreitenerweiterung versteht man die Verbreiterung der Wortbreite bei gleicher Ansteuerung der Speicherbausteine
- Beispiel 2kx32 aus vier 2kx8 Bausteinen
- Der 32-bit Datenbus wird aus vier 8-bit Teilbussen gebildet



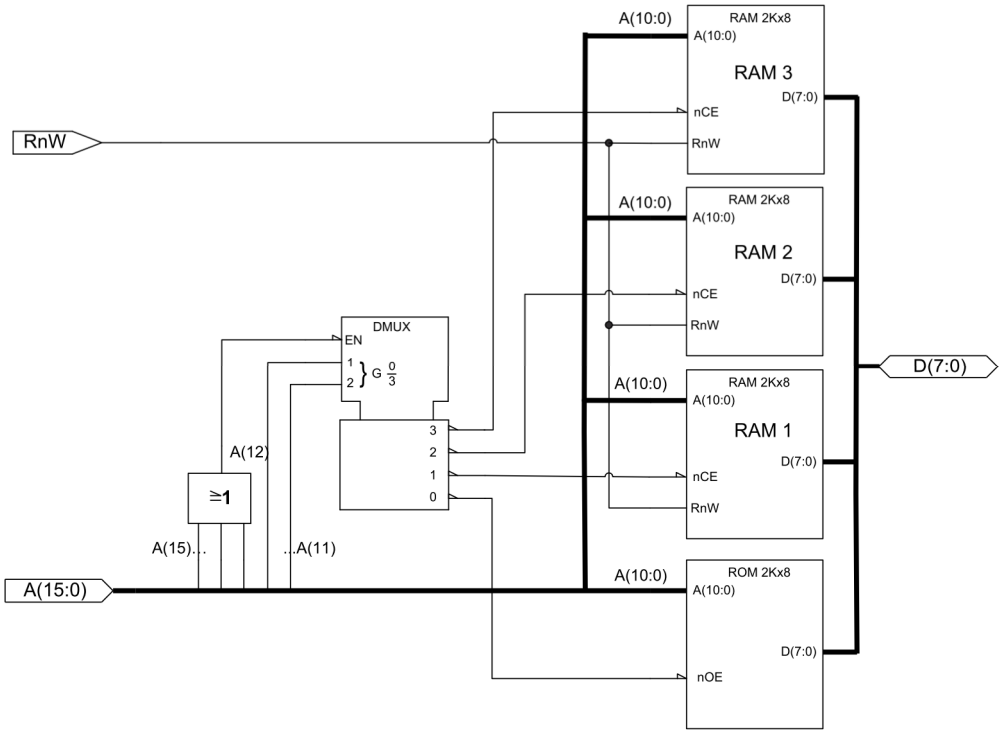
Speichersysteme - Speicherkaskadierung - Speicherkapazitätserweiterung I/III

- Bei der Speicherkapazitätserweiterung erfolgt die Aufteilung des Adressraums do, dass mehrere verschiedene Speicherbausteine aktiviert werden
- Beispiel 16bit Byte-adressierbarer Adressraum mit drei RAM mit 2kx8 und einem Rom mit 2kx8



Speichersysteme - Speicherkaskadierung - Speicherkapazitätserweiterung II/III

- Die Bausteine werden alle mit den Adressbits A(10:0) adressiert
- Die chip-enable Leitungen (nCE) werden über einen 2 zu 4 Dekoder/Demultiplexer erzeugt
- Die Adressen A12 und A11 werden dekodiert (wenn „00“ -> nCe für ROM, wenn „01“ nCE für RAM1, wenn „10“ nCE für RAM2, wenn „11 nCe für RAM3
- Damit oberhalb des verwendeten Adressbereichs kein Speicherbaustein versehentlich aktiviert wird, werden die unbenutzten Adressleitungen verodert und auf den low-aktiven enable-Eingang (EN) des Dekoders gegeben



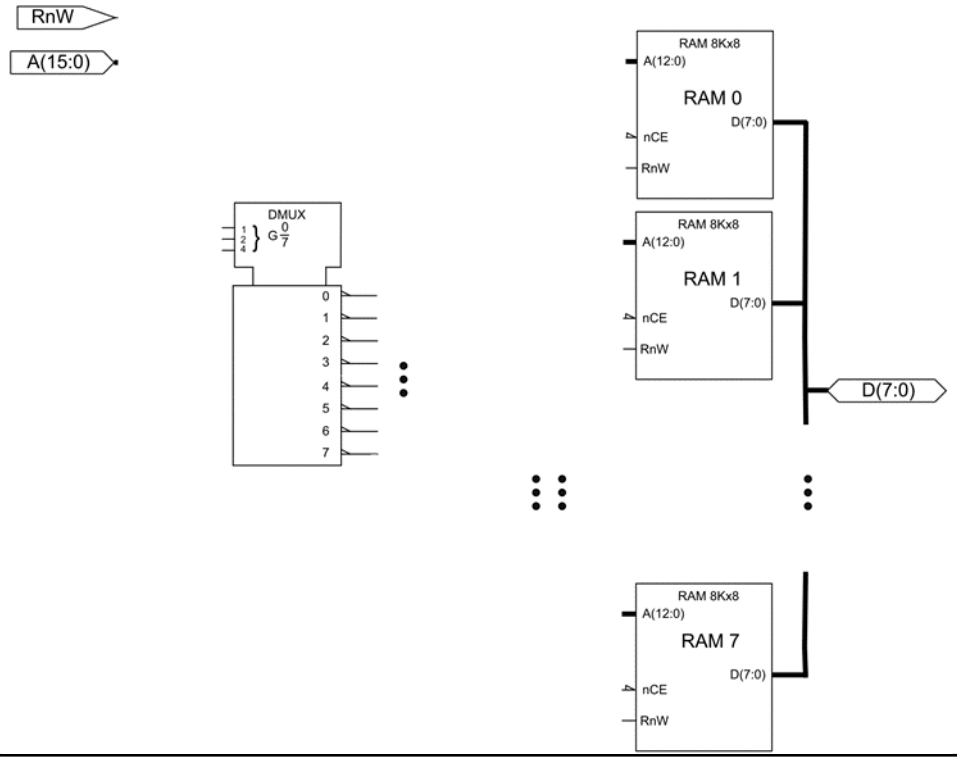
Speichersysteme - Speicherkaskadierung - Speicherkapazitätserweiterung III/III

- Diese Vorgehensweise nennt man Vollkodierung
- Wenn man das Oder-Gatter weglässt spricht man von Teilkodierung
- Es ergibt sich dabei folgende memory map:

Baustein	Adress-bereich	Binäradresse															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM	0x0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0x07FF	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
RAM 1	0x0800	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	0x0FFF	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM 2	0x1000	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	0x17FF	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
RAM 3	0x1800	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
	0x1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
nicht ver-wendet	0x2000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	0xFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

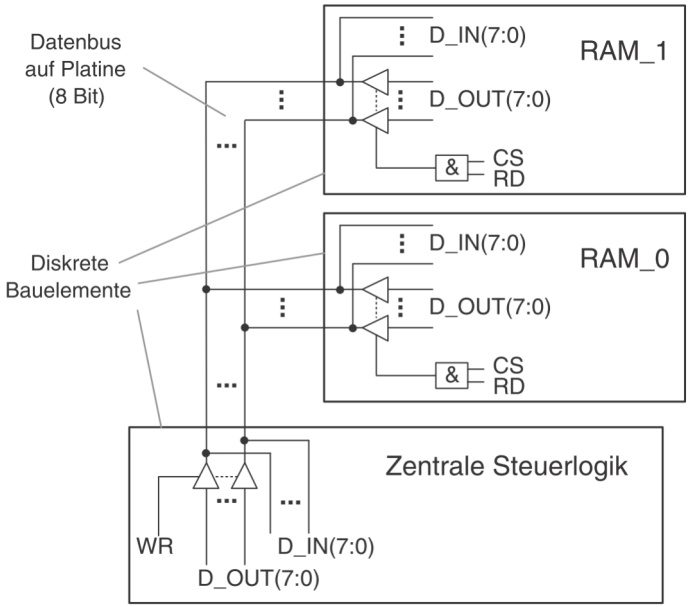
Speichersysteme - Speicherkaskadierung - Speicherkapazitätserweiterung - Übung

- In einem 64-kB Adressraum sollen acht 8kB-RAM (RAM_0 … RAM_7) angeschlossen werden
- Vervollständigen Sie die Skizze für die Schaltung für eine vollständige Decodierung (vereinfachte Skizze mit RAM0, RAM1 und RAM7 reicht)
- In welchem RAM befindet sich die Adresse 0xAFFE?



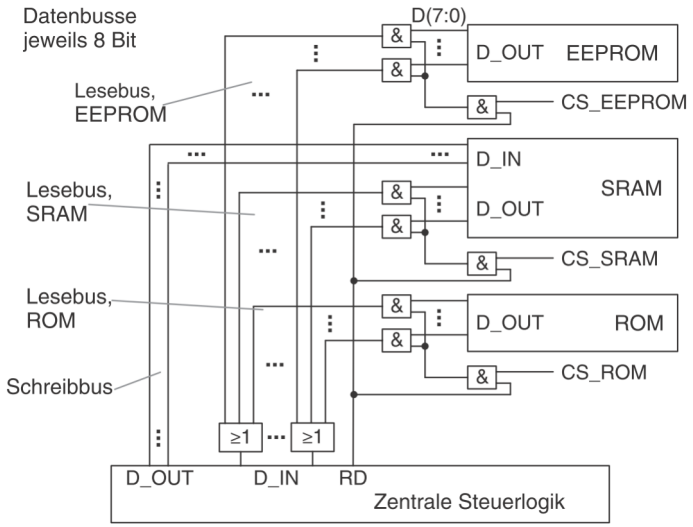
Speichersysteme - Datenbus - Tristate (R)

- Wenn externe Speicherbausteine auf einer Platine verwendet werden, so werden in der Regel Tri-State Ausgänge in den Speicherbausteinen verwendet, um sich an den Datenbus des Mikroprozessors anzuschließen



Speichersysteme - Datenbus - Multiplexing I/II (R)

- Wenn sich die Speicher on-chip befinden (z.B. FPGA-basiertes Prozessorsystem), so wird der Datenbuss in der Regel „gemultiplexed“
- Grund dafür ist, dass sich Tri-State-Treiber sehr schlecht on-chip realisieren lassen
- Beispiel Speichersystem mit EEPROM, SRAM und ROM
- Alle drei Module haben einen Datenausgang, aber nur das SRAM hat einen Dateneingang
- Der Schreibbus wird nicht „gemultiplexed“, da es nur eine Quelle gibt (nur der Prozessor kann schreiben)



Speichersysteme - Datenbus - Multiplexing II/II (R)

- Der Lesebus hat mehrere Quellen und muss daher „gemultiplexed“ werden
- Jedes Modul hat daher die RD-leitung mit der CS-Leitung UND-verknüpft
- Dieses Signal wird dann auf weiter UND-Gatter gegeben, die eine Gate-Schaltung für die Ausgänge darstellen
  - So lange dieses Signal 0 ist sind auch die durch das Gate gehende Ausgänge der Bausteine 0
- Da immer nur ein Lesebus aktiv ist, sind die anderen Lesebusse 0 und können daher über eine ODER-Verknüpfung mit dem Lesebus der Zentralen Steuerlogik verbunden werden