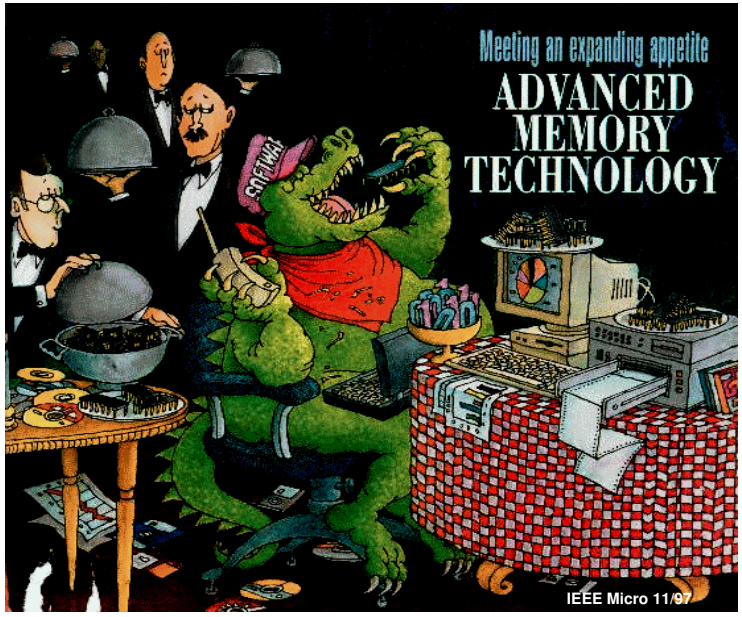
Speicher: Übersicht

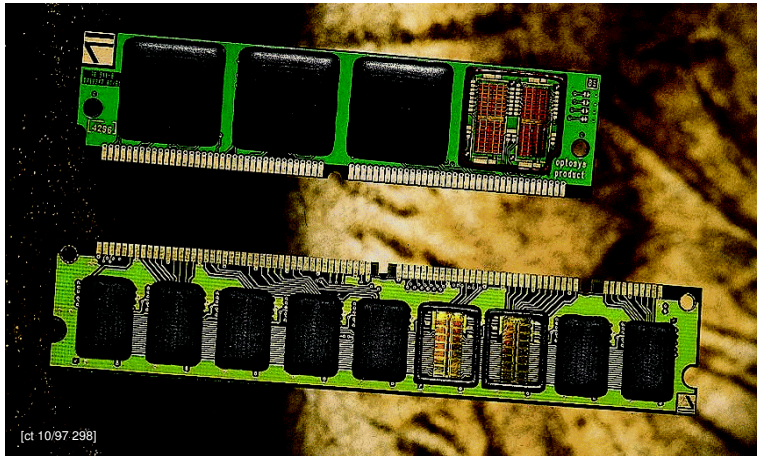
- Motivation: "performance gap" zwischen CPU und Speicher
- DRAM Grundlagen
- Speicherhierarchie, Cache
- SDRAM, Rambus
- IRAM

Speicher: Literatur

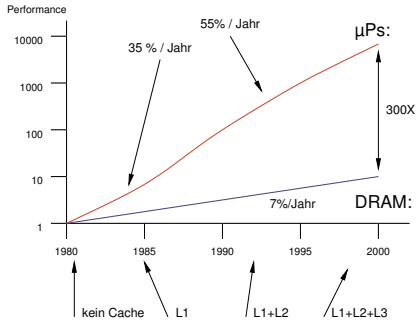
[IEEE Micro 3/97]	IRAM
[IEEE Micro11/97]	Advanced Memory Technology: Übersicht, RAMBUS, SLDRAM
[Hennessy & Patterson]	Kapitel 5, Speicherhierarchie
[c't 07/96 p.158]	"SIMMsalabim"
[c't 10/97 p.298]	"Schnelle Speicherkäfer"
[c't 96-2000]	diverse Testberichte
www.rambus.com	alle RAMBUS Docs
www.jedec.org	Standards
developer.intel.com	Memory homepage, Chipsätze
[[Cvetanivic/Bhandarkar ISCA 96]	Performance-Analyse Alpha-21164



SIMM / DIMM: 72/168 polig 32/64 bit



DRAM: Performance Gap



- DRAM-Kapazität: 60% / Jahr, Latenz: 7% / Jahr
- Prozessor-Performance: 55% / Jahr
- Kluft vergrößert sich ständig

=> Speicherhierarchie mit Caches notwendig

DRAM: Performance gap: Beispiel

- Zeit für L2-Cache-Miss (# idle instructions):

Alpha 21064 (7000):	340ns / 5.0ns	68 clocks x 2 = 136
Alpha 21164 (8400):	266ns / 3.3ns	80 clocks x 4 = 320
Alpha 21264 (est.):	180ns / 1.7ns	108 clocks x 6 = 648

. . .

- Caches essentiell notwendig, um DRAM-Latenz zu verstecken
- Problem wird mit jeder Prozessorgeneration schlimmer
- Beispiel: Analyse für Alpha 21164 [ISCA'96]

- CPU mit idealem Speicher:
  Performance durch Verlustleistung limitiert (ca. 50Watt)

DRAM: Performance Gap: Was tun?

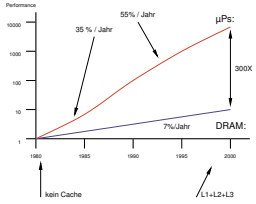
schnellerer Speicher notwendig ...

- aber DRAM inhärent langsam
- SRAM sehr teuer

=> DRAM besser ausnutzen
- SDRAM, SDRAM-DDR
- RAMBUS, SLDRAM

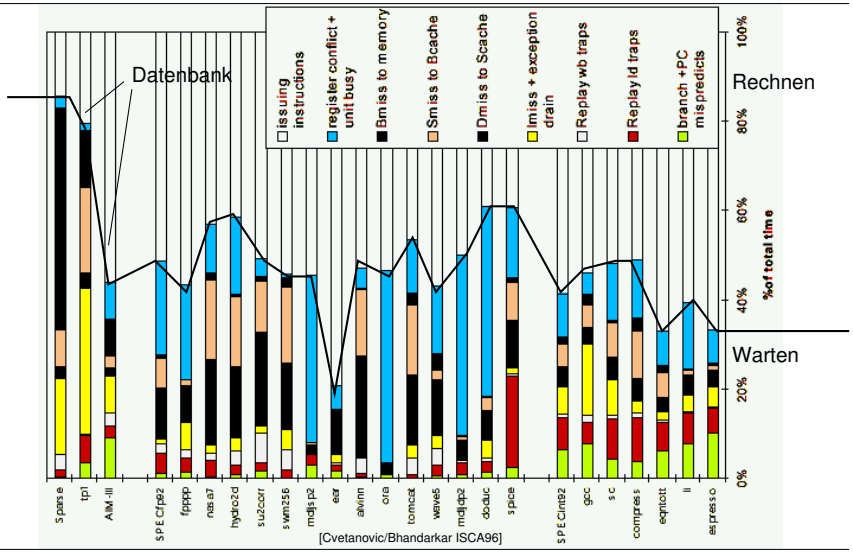
=> Speicherhierarchie
- größere, schnellere Caches
- bessere Cache-Organisation
- Prefetch-Optimierungen

=> neue Konzepte?
- IRAM



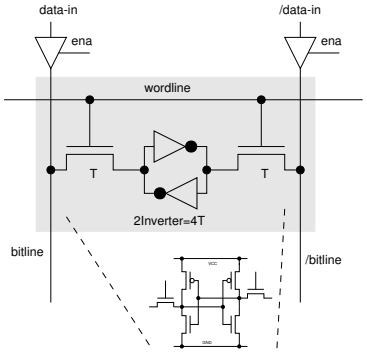
"Cache: a safe place for hiding or storing things"
Websters dictionary

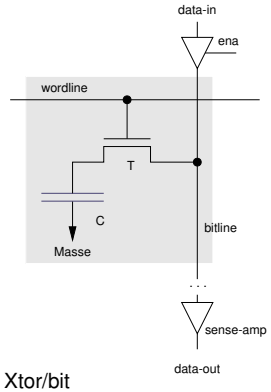
DRAM: Alpha 21164



[Cvetanovic/Bhandarkar ISCA96]

DRAM vs. SRAM

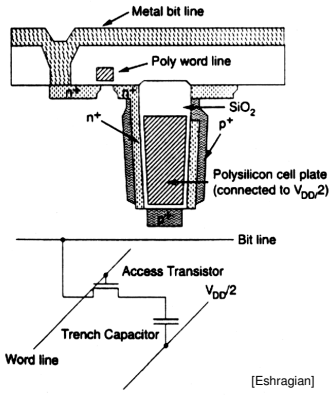




- 6 Xtors/bit
- statisch (kein refresh)
- schnell
- 10 .. 50X DRAM-Fläche

- 1 Xtor/bit
- C=10fF: ~200.000 Elektronen
- langsam (charge-sharing)
- minimale Fläche

DRAM: Trench-Kondensator



[Eshragian]

- Bauform Trench: Platten vertikal am Rand eines Grabens
- Bauform Stack: mehrere horizontale Schichten

DRAM: Stack / Trench-Kondensator

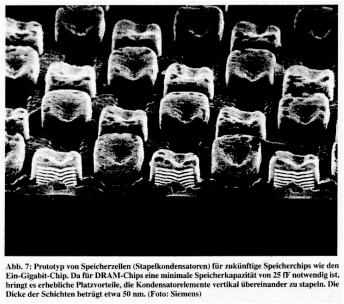
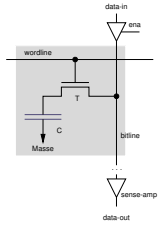


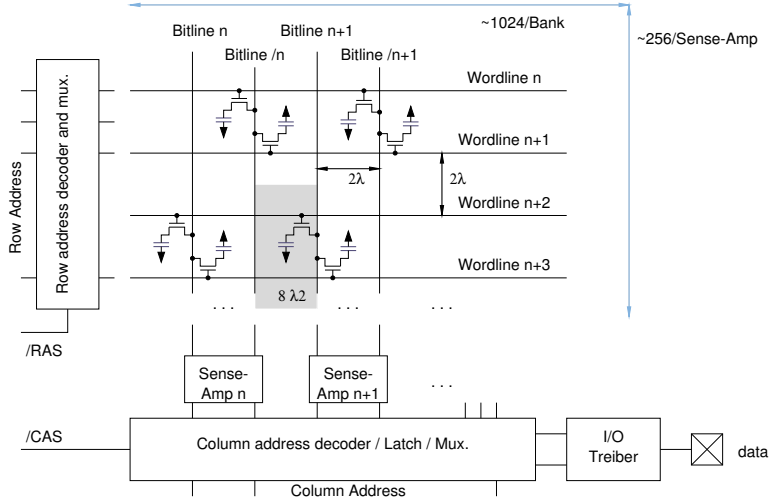
Abb. 7: Prototyp von Speicherzellen (Stapelkondensatoren) für zukünftige Speicherchips wie den Ein-Gigabit-Chip. Da für DRAM-Chips eine minimale Speicherkapazität von 25 fF notwendig ist, bringt es erhebliche Platzvorteile, die Kondensatorelemente vertikal übereinander zu stapeln. Die Dicke der Schichten beträgt etwa 50 nm. (Foto: Siemens)



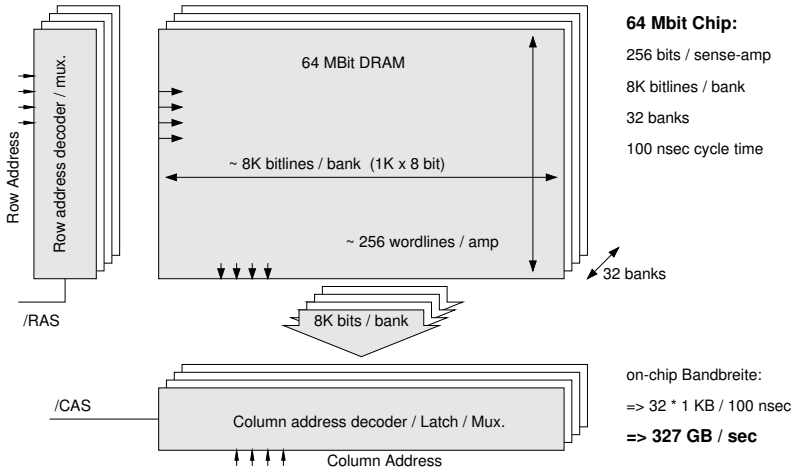
- "stacked capacitors"
  [Siemens 1Gb DRAM Prototyp 96]
- C=10fF: ~200.000 Elektronen

- "trench capacitors"
  [IBM CMOS-6X embedded DRAM]

DRAM: Layout



DRAM: Organisation / Bandbreite



64 Mbit Chip:

256 bits / sense-amp

8K bitlines / bank

32 banks

100 nsec cycle time

on-chip Bandbreite:

=> 32 * 1 KB / 100 nsec

=> 327 GB / sec

DRAM: Funktion

Read:

- /RAS = 0: Auswahl der Wordline,Aktivierung der Bitlines
  Auslesen und Auswertung der selektierten Zellen
- /CAS = 0: Auswahl der Bitline, Ausgabe der Daten
  Zurückschreiben der gelesenen Daten (!)
- /RAS = 1: Precharge der Bitlines

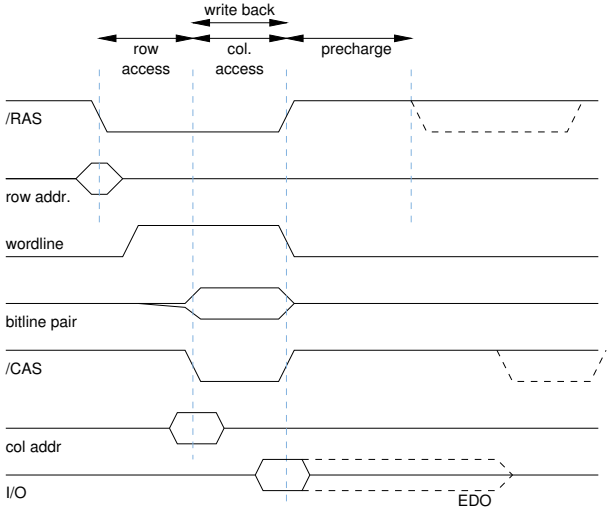
Write:

- /CAS = 0: Zurückschreiben der gelesenen + neuer Daten

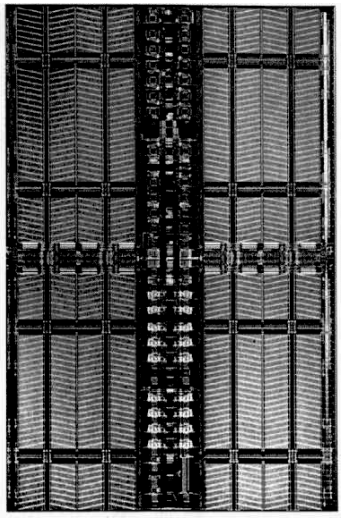
SDRAM: zusätzliche Register, diverse Burst-Modi

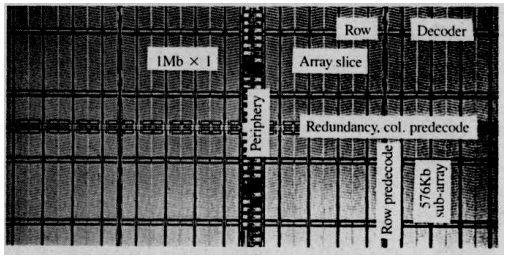
Refresh: alle 16 .. 32 ms notwendig

DRAM: Ansteuerung (asynchron)



DRAM: Floorplan (IBM 4Mbit)





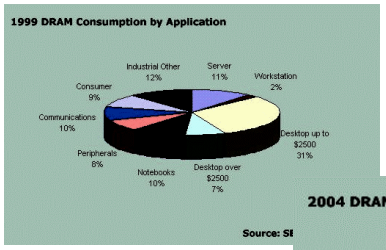
- Größenvergleich zwischen I/O, Col/Row-Decoder, Array
- Konfiguration nach Marktlage
  links: 4 Mbit, oben: 16 Mbit
- Redundanz für besseren Yield:
  links: 4.0/4.5 Mbit Kapazität/brutto
  [IBM JR&D 1995]

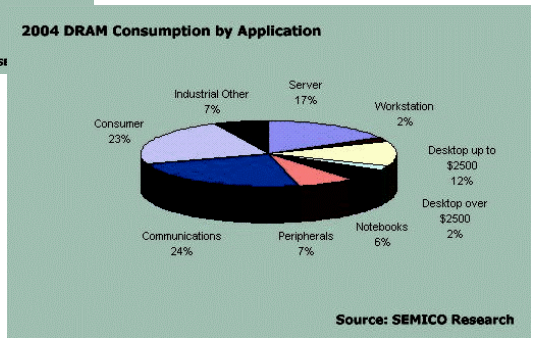
DRAM: Trend und Dilemma

- Preisverfall: 16Mb: 50$ @ 1/96 -> 10$ @ 12/96 -> 4$ @ 12/97
- Anzahl DRAMs / Computer sinkt:
  - Kapazität steigt mit 50% - 60% / Jahr
  - Software benötigt 33% / Jahr
  - Mindestanzahl gegeben durch Busbreite vs. DRAM-Breite (4bit)
- überhaupt ein Markt für große DRAMs? (256Mb, 1Gb, ...)

# Chips	'86 1Mb	'89 4Mb	'92 16Mb	'96 64Mb	'99 256Mb	'02 1Gb
4 MB	32 →	8			60% / Jahr →	
8 MB		16 →	4			
16 MB			8 →	2		
32 MB			16 →	4 →	1	
64 MB				8 →	2	
128 MB	33% / Jahr				4 →	1
256 MB	↓				8 →	2

DRAM: Halbleitermarkt





- PC verliert Anteile
\+ Consumer-Apps
\+ Netzwerke

DRAM: der Halbleitermarkt

DRAM als Standardbauteile: erfordert standardisierte Schnittstelle

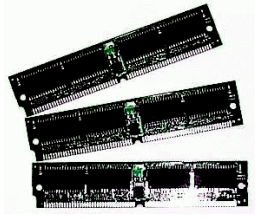
- Markt (1995): DRAMs 37 Mrd. $, µPs 20 Mrd. $

- hohe Stückzahlen, viele Lieferanten, wenig Profit
- 'quadratische' Speichermatrix mit N*N Bits, extern 1/4/16 Bits
- Architekturverbesserungen minimal: PM, EDO, SDRAM, DDR, ...
- Generationen: 64 Kb, 256 Kb, 1Mb, 4Mb, 16Mb, 256Mb, ... (1 Gb)

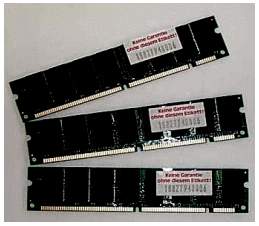
- "kleine" Anwendungen müssen bestehendes Angebot nutzen
- spezielle Varianten bei entsprechender Stückzahl (z.B. N64, PSX2)
- PC-Markt bestimmt die Marschrichtung

- Integration von DRAM und Logik zunehmend aktuell (IRAM &Co)

DRAM: Bauformen SIMM / DIMM / RIMM



EDO-SIMM 60ns. 72p.



SDRAM-100 DIMM 168p.

RAMBUS-PC800 RIMM 168p

SDRAM:

SDRAM: synchrone Ansteuerung für bessere Performance:

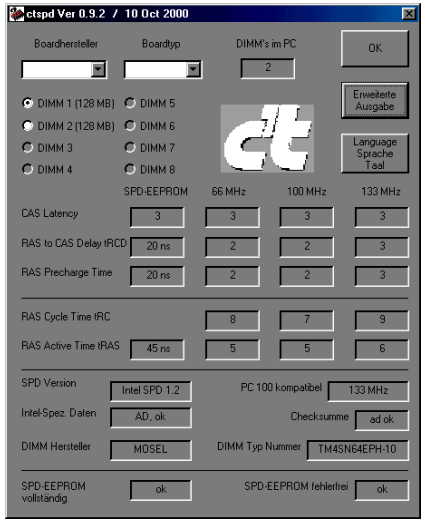
- interner Aufbau wie asynchrone DRAMs

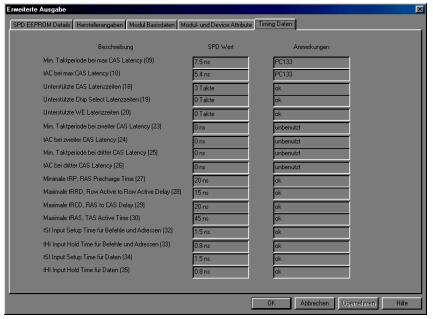
- getaktete I/O-Register
- Wertekombination auf CD/nRD/nWE/... wird als Befehl interpretiert
- mehrere Burst Read/Write Modi
- Mode-Register, etwa Auswahl Burstlength 1/2/4/8
- übliche Taktraten 66 MHz / 100 MHz / 133 MHz

- PC-66 / und PC-100 Spezifikationen von Intel
- PC-133 Spezifikation zuerst von VIA / von Intel übernommen
- diverse Varianten (SGRAM / double data rate "DDR" / ...)
- Marktbedeutung <=> Patentstreitigkeiten (u.a. mit Rambus, Inc.)

[developer.intel.com/memory]

SDRAM: SPD EEPROM Daten





"serial presence detect":

- EEPROM mit allen Timing-Daten
- volle Autokonfiguration
- typ. Zeiten 20 .. 50 nsec.

SDRAM: Commands

Command Truth Table

Function	Symbol	CKE n-1	CKE n	CS#	RAS#	CAS#	WE#	A11	A10	BA(0:1)	A9-A0
Device deselect	DSEL	H	X	H	X	X	X	X	X	X	X
No Operation	NOP	H	X	L	H	H	H	X	X	X	X
Read	READ	H	X	L	H	L	H	V	L	V	V
Read w/ auto precharge	READAP	H	X	L	H	L	H	V	H	V	V
Write	WRIT	H	X	L	H	L	L	V	L	V	V
Write w/ auto precharge	WRITEAP	H	X	L	H	L	L	V	H	V	V
Bank Activate	ACT	H	X	L	L	H	H	V	V	V	V
Precharge select bank	PRE	H	X	L	L	H	L	V	L	V	X
Precharge all banks	PALL	H	X	L	L	H	L	X	H	X	X
Auto refresh	CBR	H	H	L	L	L	H	X	X	X	X
Self refresh entry from IDLE	SLFRSH	H	L	L	L	L	H	X	X	X	X
Self refresh exit	SLFRSHX	L	H	H	X	X	X	X	X	X	X
Power Down entry from IDLE	PWRDN	H	L	X	X	X	X	X	X	X	X
Power Down exit	PWRDNX	L	H	H	X	X	X	X	X	X	X
Mode register set	MRS	H	X	L	L	L	L	L	L	V	V

steigende Taktflanke:

- nCS
- nRAS
- nCAS
- nWE

=> SDRAM-Befehl

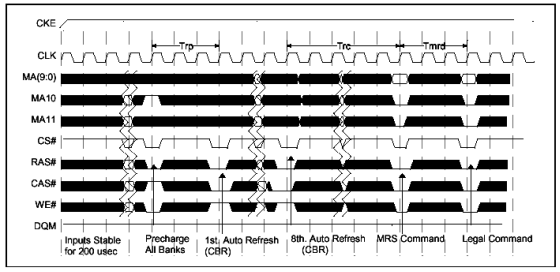
Leerseite

SDRAM: Initialisierung

The initialization sequence can be issued at *anytime*. Following the initialization sequence, the device must be ready for full functionality. SDRAM devices are initialized by the following sequence:

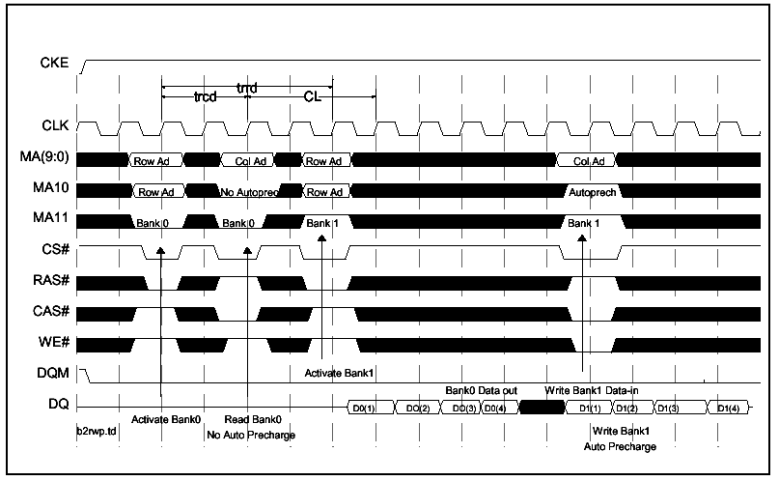
1. At least one NOP cycle will be issued after the 1msec device deselect.
2. A minimum pause of 200usec will be provided after the NOP.
3. A precharge all (PALL) will be issued to the SDRAM.
4. 8 Auto refresh (CBR) refresh cycles will be provided.
5. A mode register set (MRS) cycle will be issued to program the SDRAM parameters (e.g., Burst length, CAS# latency etc.).
6. After MRS the device should be ready for full functionality within 3 clocks after T_{mrd} is met.

Power Up Initialization Sequence



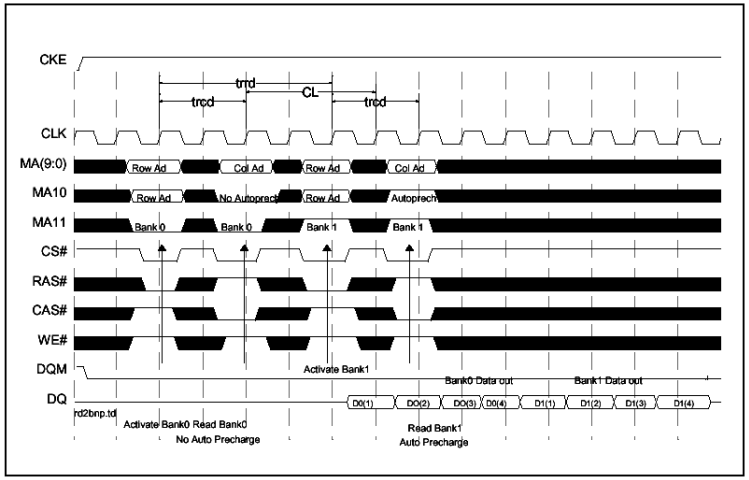
SDRAM: Read / Write Bursts

Read and Write Commands (Burst Length 4 Shown)



SDRAM: "Ping Pong Read"

Two Bank Ping Pong Read



Leerseite

SDRAM: DDR Read

Burst Read Operation

The burst read operation in DDR SDRAM is done in the same manner as the current SDRAM. The burst read command is issued by asserting $\overline{CS}$ and $\overline{CAS}$ Low while holding $\overline{RAS}$ and $\overline{WE}$ High at the rising edge of the clock (CLK) after T_{RCD} from the bank activation. The address inputs determine the starting address for the burst. The mode register sets the type of burst (sequential or interleave) and the burst length (2, 4, 8). The first output data is available after the $\overline{CAS}$ latency from the read command, and the consecutive data are presented on the falling and rising edge of DQS until the burst length is completed.

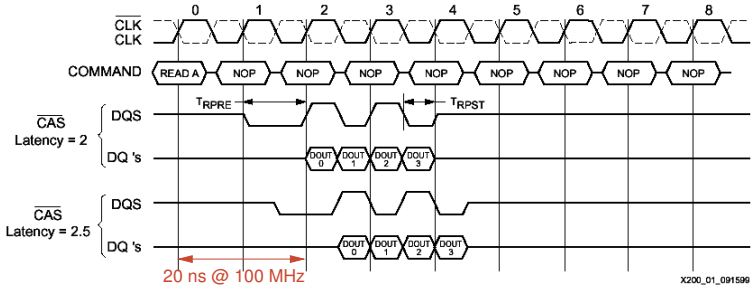


Figure 1: **Burst Read Operation Timing**

[Xilinx appnote]

SDRAM: DDR Write

Burst Write Operation

The burst write command is issued by having $\overline{CS}$, $\overline{CAS}$ and $\overline{WE}$ Low while holding $\overline{RAS}$ High at the rising edge of the clock (CLK). The address inputs determine the starting column address. There is no write latency relative to DQS required for burst write cycle. The first data of a burst write cycle must be applied on the DQ pins T_{DS} (data-in setup time) prior to data strobe edge. The data strobe signal is enabled after T_{DQSS} from the rising edge of CLK issued by the WRITE command. The remaining data inputs must be supplied on each subsequent falling and rising edge of Data Strobe until the burst length is completed. When the burst has been finished, any additional data supplied to the DQ pins will be ignored.

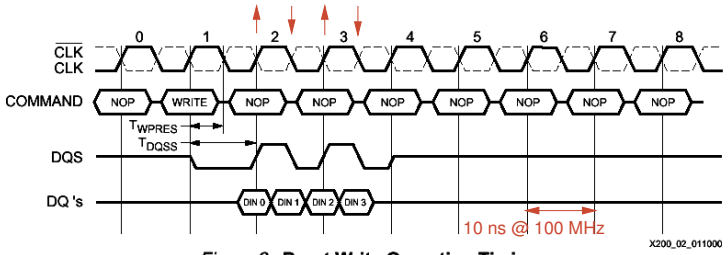


Figure 2: **Burst Write Operation Timing**

[Xilinx appnote]

SDRAM: DDR Controller

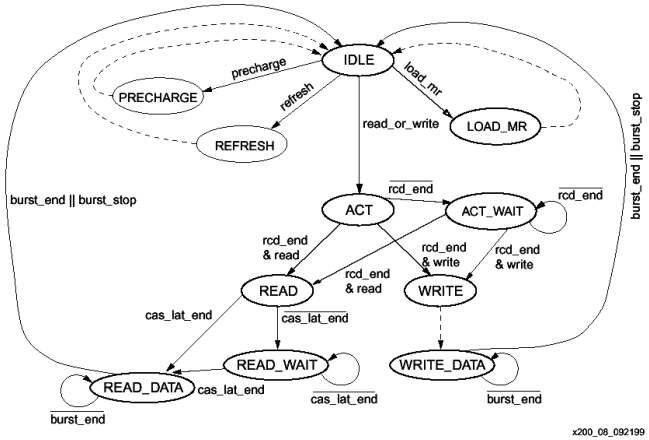


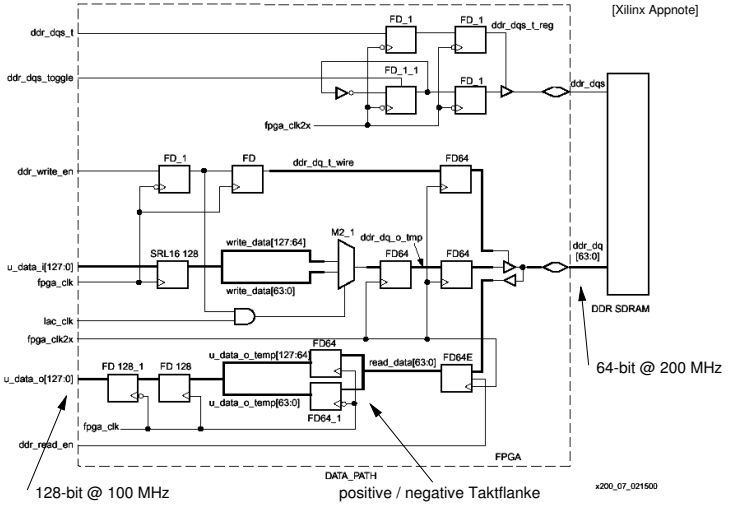
Figure 8: **State Machine Diagram**

An overview of the State Machine is shown in Figure 8. The dashed lines indicate an automatic sequence.

[Xilinx appnote]

SDRAM: DDR Datenpfad

[Xilinx Appnote]



RAMBUS: Motivation

- steigende Anforderungen (etwa für 3D-Apps.)
- immer mehr Speicherbandbreite erforderlich
- sinkende Anzahl einzelner DRAM-Chips

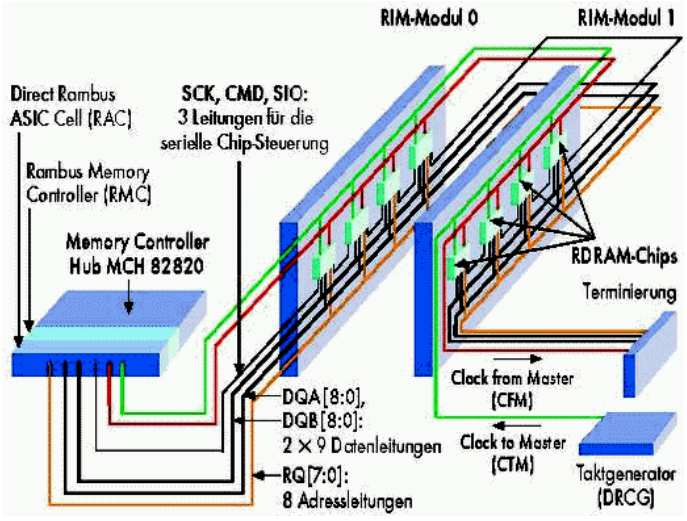
- Bustakt (133 MHz) kaum weiter zu steigern
- breitere Busse als 64 bit sehr teuer
- Boards sollen minimale/maximale Bestückung vertragen
- DDR problematisch, da Verzögerungen bereits ausgereizt

=> konventionelle Speichertechnik "am Anschlag"

=> RAMBUS

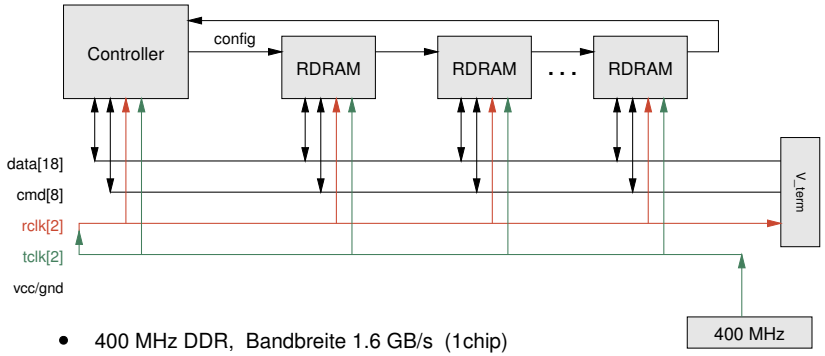
- timing-optimierter Bus (266 .. 400 MHz DDR)
- wenige Leitungen (18 data + 8 cmd + 4 clock + vcc + gnd)
- flexible Bestückung (N64/PSX2: nur je 2 Chips)

Rambus: Prinzip



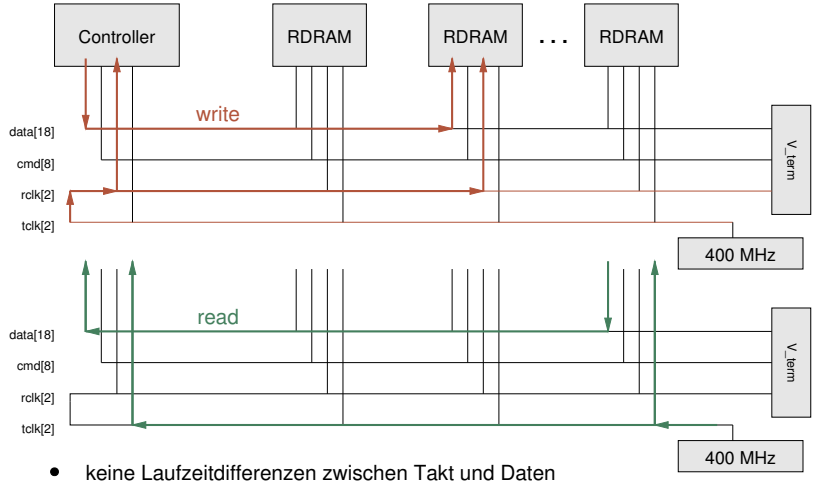
[ct 03/2000]

RAMBUS: Konzept



- 400 MHz DDR, Bandbreite 1.6 GB/s (1chip)
- 8-bit Adressen, 16+2 bit Daten
- gespiegelte Taktleitungen transmit/return für Read/Write (!)
- chipintern 128/144 bit @ 10 nsec
- flexibel: Timing angepaßt an Anzahl / Lage der Chips

RAMBUS: Read/Write



- keine Laufzeitdifferenzen zwischen Takt und Daten
- Zugriff auf hintere Chips ist langsamer

RAMBUS: signal delay matching

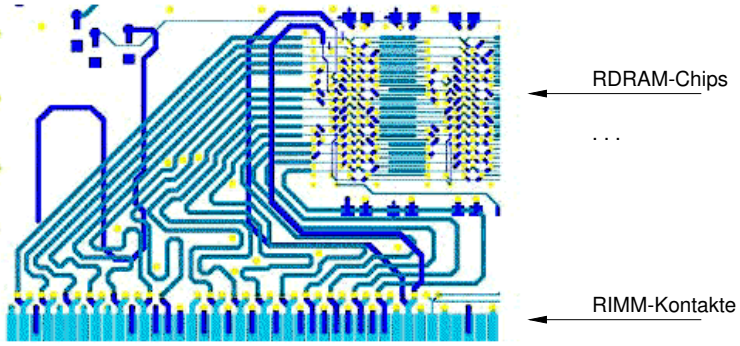


Figure 5-10: Delay Matching (Left Side)

- Leitungslängen angepasst für einheitliche Laufzeiten
- 800 MHz / 1.25 nsec / ~ 18 cm 0.5 nsec / ~ 7 cm

RAMBUS: basic read / write transactions

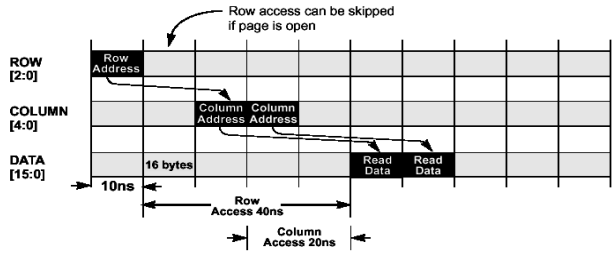


Figure 6: Read Transaction

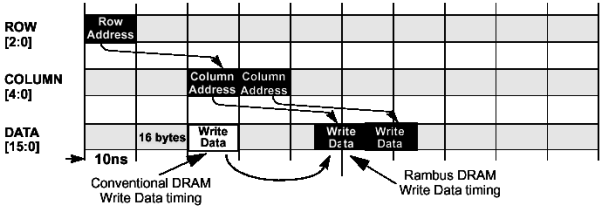
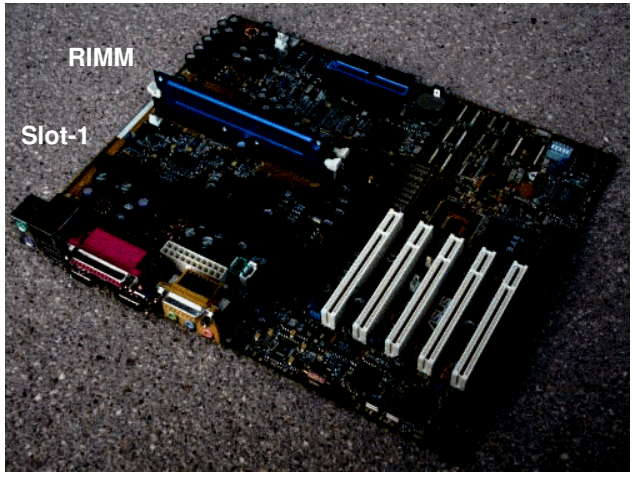


Figure 7: Write Transaction

RAMBUS: Asus P3C (i820)



RAMBUS: basic read / write transactions

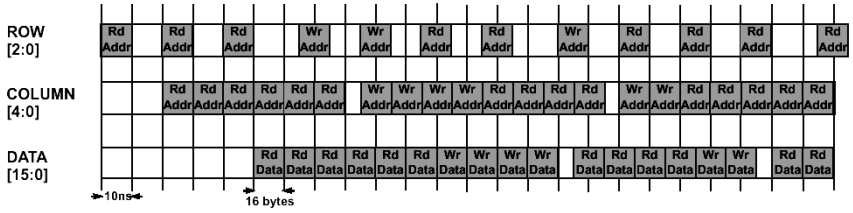
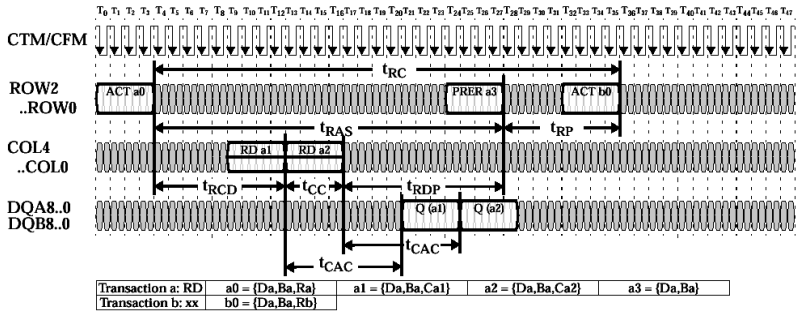


Figure 8: Simultaneous Pipelined Transaction

- separate Steuerleitungen für Row / Column-Select
- ermöglicht Pipelining von Lese- und Schreibzugriffen

- Datenleitungen im Idealfall fast 100 % ausgelastet
- aber nur mit geeigneten Zugriffen (32-Byte Ausrichtung)
- Performance Compiler-abhängig

RAMBUS: Read

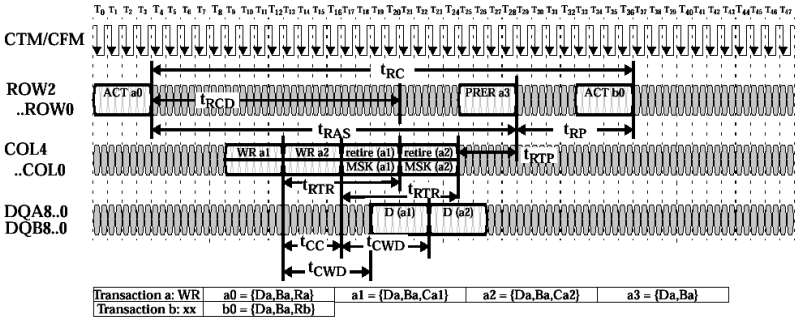


Transaction a: RD	a0 = {Da,Ba,Ra}	a1 = {Da,Ba,Ca1}	a2 = {Da,Ba,Ca2}	a3 = {Da,Ba}
Transaction b: xx	b0 = {Da,Ba,Rb}			

Figure 15: Read Transaction Example

- tRCD / tCC / tCAC abhängig vom Modul (PC-800 / PC-700 / usw.)
- zusätzliche Latenztakte für "hintere" Module
- zusätzliche Latenztakte zur Temperaturregelung
- (1 Chip reicht für volle Datenrate => höhere Belastung als bei SDRAM)

RAMBUS: Write

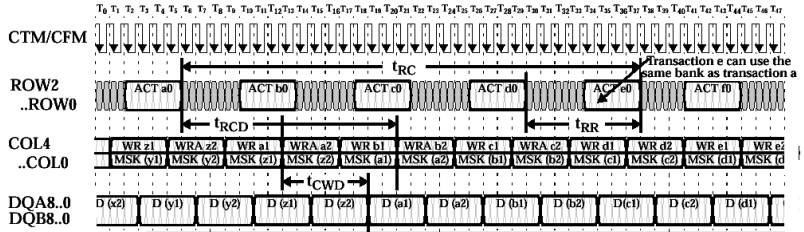


Transaction a: WR	a0 = {Da,Ba,Ra}	a1 = {Da,Ba,Ca1}	a2 = {Da,Ba,Ca2}	a3 = {Da,Ba}
Transaction b: xx	b0 = {Da,Ba,Rb}			

Figure 16: Write Transaction Example

- tRCD / tCC / tCAC abhängig vom Modul (PC-800 / PC-700 / usw.)
- zusätzliche Latenztakte für "hintere" Module
- zusätzliche Latenztakte zur Temperaturregelung
- (1 Chip reicht für volle Datenrate => höhere Belastung als bei SDRAM)

RAMBUS: Interleaved Write

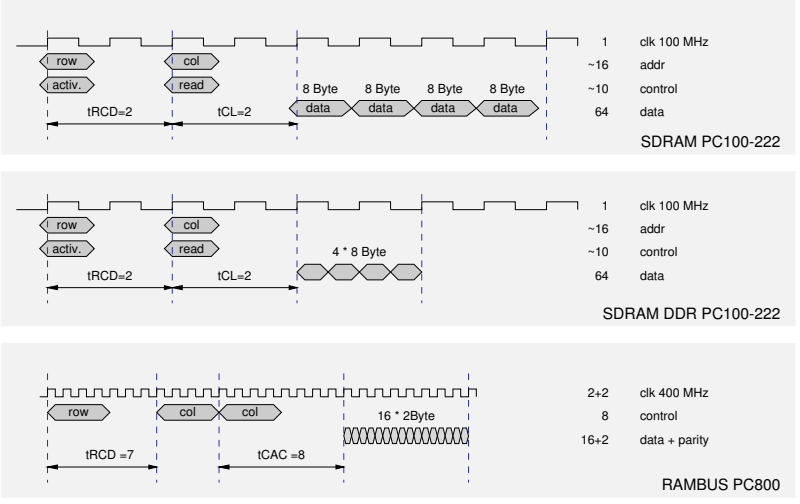


Transaction y: WR	y0 = {Da,Ba+4,Ry}	y1 = {Da,Ba+4,Cy1}	y2= {Da,Ba+4,Cy2}	y3 = {Da,Ba+4}
Transaction z: WR	z0 = {Da,Ba+6,Rz}	z1 = {Da,Ba+6,Cz1}	z2= {Da,Ba+6,Cz2}	z3 = {Da,Ba+6}
Transaction a: WR	a0 = {Da,Ba,Ra}	a1 = {Da,Ba,Ca1}	a2= {Da,Ba,Ca2}	a3 = {Da,Ba}
Transaction b: WR	b0 = {Da,Ba+2,Rb}	b1 = {Da,Ba+2,Cb1}	b2= {Da,Ba+2,Cb2}	b3 = {Da,Ba+2}
Transaction c: WR	c0 = {Da,Ba+4,Rc}	c1 = {Da,Ba+4,Cc1}	c2= {Da,Ba+4,Cc2}	c3 = {Da,Ba+4}
Transaction d: WR	d0 = {Da,Ba+6,Rd}	d1 = {Da,Ba+6,Cd1}	d2= {Da,Ba+6,Cd2}	d3 = {Da,Ba+6}
Transaction e: WR	e0 = {Da,Ba,Re}	e1 = {Da,Ba,Ce1}	e2= {Da,Ba,Ce2}	e3 = {Da,Ba}
Transaction f: WR	f0 = {Da,Ba+2,Rf}	f1 = {Da,Ba+2,Cf1}	f2= {Da,Ba+2,Cf2}	f3 = {Da,Ba+2}

Figure 20: Interleaved Write Transaction with Two Dualoct Data Length

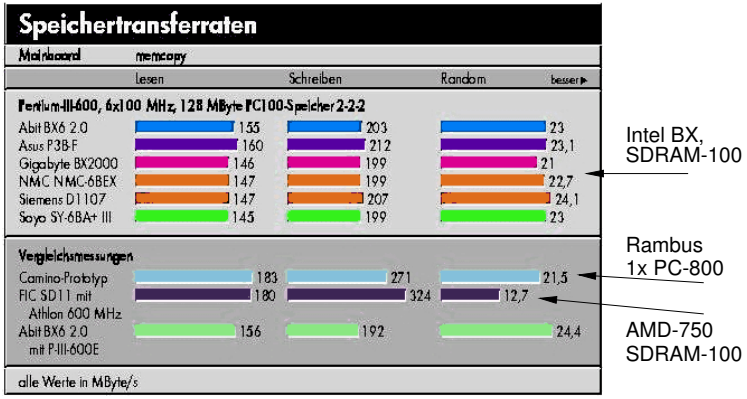
- entsprechend komplexe Zyklen auch für Read
- zusätzliche Buszyklen für Refresh / Powermanagement / usw.

RAMBUS: vs. SDRAM / SDRAM-DDR



[c't 04/00 S.232 / rambus.com]

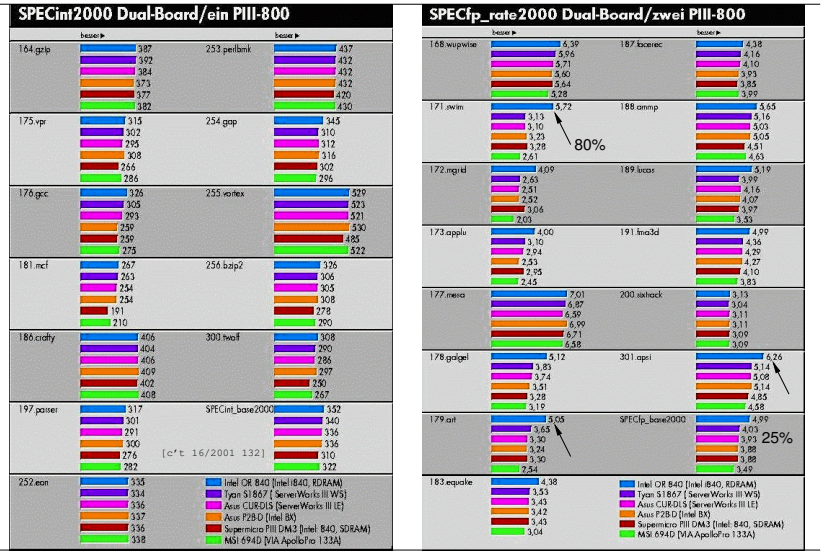
RAMBUS: c't memcopy



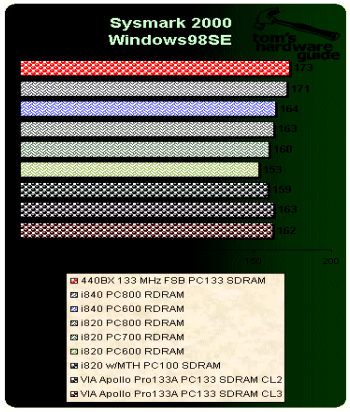
- starke Streuung selbst bei gleichem Chipsatz
- single-channel RAMBUS nicht überragend

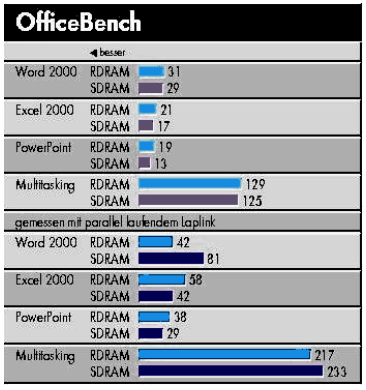
[c't 24/99/118]

RAMBUS: SPECint 2000 / SPECfp 2000



RAMBUS: Office Benchmark





- Speicher überhaupt gefordert ?!
- SDRAM 133 besser als RDRAM ?!

[www.tomshardware.com]
[c't / www.dell.com]

RAMBUS: Fazit . . .

lohnt die neue, teure Technik?

- interessantes und flexibles Konzept
- ein Chip reicht für volle Datenrate: geeignet für 1Gbit Generation
- volle Autokonfiguration und adaptives Bustiming

- widersprüchliche Benchmark-Ergebnisse
- single-channel RDRAM-800 kaum besser als SDRAM-133
- dual-channel RDRAM-800 teuer aber gut
- Rolle von SDRAM-DDR ?!

- derzeit nur Markenmodule, keine "no name" Billigware
- Preise bisher nicht konkurrenzfähig

- neueste Intel-Roadmap "unklar" (developer forum, Feb'00)
  RDRAM (desktop) + SDRAM (mobile, server) + advanced DRAM

Cache: Motivation

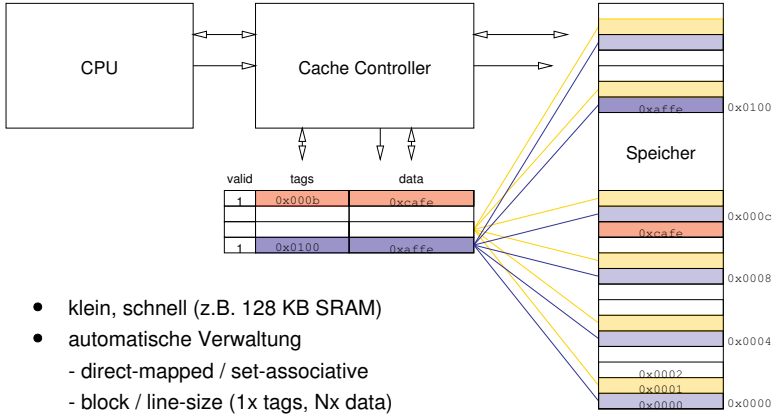
- DRAM langsam, SRAM teuer
- Lokalität: Daten mehrfach genutzt / benachbarte Daten genutzt

=> "Cache"

- kleiner SRAM-Zwischenspeicher
- Cache-Treffer laufen mit SRAM-Performance
- aber Overhead: Misses langsamer als ohne Cache

Parameter:	Beispiel
• Grösse	64 KByte
• hit-time	1 clk
• miss-time	50 clk
• miss-rate	1%
• Organisation	voll-assoziativ

Cache: Prinzip



- klein, schnell (z.B. 128 KB SRAM)
- automatische Verwaltung
  - direct-mapped / set-associative
  - block / line-size (1x tags, Nx data)
  - write-through / -back / -allocate
- Vergleich der Tags, abhängig davon Cache- oder Speicherzugriff

Cache: Parameter . . .

- Gesamtgröße, Blockgröße, Zugriffszeit, Miss-Zeit, ...

• Organisation:	wo kann ein Block platziert werden?	(direct-mapped)
• Zugriff:	wie wird ein Block gefunden?	(tags, valid bit)
• Ersetzung:	welcher Block wird beim Miss ersetzt?	(random, LRU)
• Schreib-Strategie:	write back / write through / ...	(dirty bit, ...)
• Architektur:	separate I/D oder unified Cache?	
• größere Blocks	(weniger Verdrängung, aber geringere Kapazität)	
• höhere Assoziativität	(aber komplexere Verwaltung)	
• Victim-Caches	(billig und effizient)	
• HW-Prefetching	(z.B. instruction prefetch / branch prediction)	
• Compiler-Prefetching	(bei bekannten Zugriffsmustern)	
• critical word first	(x86: von Intel patentiert)	
• write buffer	(alle aktuellen Prozessoren)	
• nonblocking caches	(effizient, aber komplex)	

Cache: AlphaServer 8200 (300MHz 21164)

memory type	size	location	latency		bandwidth
	KB		ns	cycles	MB / s
I-Cache	8K	on chip	6.6	2	4800
D-Cache	8K	on chip	6.6	2	4800
L2-Cache	96K	on chip	20.0	6	4800
L3-Cache	~ 4M	off chip	26.0	8	960
main memory	64M .. 4G	off chip	253.0	76	1200
single DRAM	16M	off chip	~ 60.0	18	30..100

		misses / 1000 instr.				% time spent in				
Program	CPI	I	D	L2	L3	µP	I	D	L2	L3
SPECint92	1.2	7	25	11	0	0.78	0.03	0.13	0.05	0.00
SPECfp92	1.2	2	47	12	0	0.68	0.01	0.23	0.06	0.02
database	3.6	97	82	119	13	0.23	0.16	0.14	0.20	0.27
sparse	3.0	0	38	36	23	0.27	0.00	0.08	0.07	0.58

[Patterson 97]

Cache: Missrate

Missrate Beispiele: (SPEC 92, R2000, direct-mapped, 32-byte blocks)

Size	Instruction	Data	Unified
1K	3.06%	24.61%	13.34%
2K	2.26%	20.57%	9,78%
4K	1.78%	15.94%	7.24%
8K	1.10%	10.19%	4.57%
16K	0.64%	6.47%	2.87%
32K	0.39%	4.82%	1.99%
64K	0.15%	3.77%	1.35%
128K	0.02%	2.88%	0.95%

- Werte sehr stark programmabhängig
- CPU / Multiuser-Auslastung / Messzeit / ...

[H&P p.384]

Cache: Compulsory / Capacity / Conflict

3 Arten Cache-Misses:

- compulsory (cold start / first reference)
  erster Zugriff auf einen Block
- capacity Cache zu klein für alle benötigten Blöcke;
  Blöcke müssen ausgetauscht werden
  => Cache vergrössern
- conflict (collision misses / interference misses)
  bei direct mapped / set associative Caches:
  mehrere Blöcke im gleichen Set benötigt
  => Organisation verbessern, etwa 4fach assoz.
  => victim buffers

Cache: Missrate: Beispiel

- Speicherzugriffe: 75% Instruction, 25% Data
- avg. memory access time
  = hit time + (miss rate * miss penalty)
- cache hit: 1 clock
- cache miss: 50 clocks

Size	Instruction	Data	Unified
1K	3.06%	24.61%	13.34%
2K	2.26%	20.57%	9,78%
4K	1.78%	15.94%	7.24%
8K	1.10%	10.19%	4.57%
16K	0.64%	6.47%	2.87%
32K	0.39%	4.82%	1.99%
64K	0.15%	3.77%	1.35%
128K	0.02%	2.88%	0.95%

16K I + 16K D Cache:

- miss rate: (75% * 0.64%) + (25% * 6.47%) = 2.10%
- tmac: 75%*(1+0.64%*50) + 25%*(1+6.47%*50) =
  (75%*1.32) + (25%*4.235) = 2.05

32K unified Cache: load/store hit: 1 extra cycle (one port only)

- miss rate: 1.99%
- tmac: 75%*(1+1.99%*50) + 25%*(1+1+1.99%*50) =
  (75%*1.995) + (25%*2.995) = 2.24

[H&P p.385]

=> split I/D Cache ist schneller (für dieses Beispiel)

Cache: direct-mapped conflict misses

```
static void filterF(char* in1, char* out1)
{
    register int i0,i1,i2;
    register int x, int y;
    register char *in,*out;
    in = in1;
    out = out1;
    for( y=0; y < YRES; y++ ) {
        i0 = (int)in[0];
        i1 = (int)in[1];
        /* ignore boundary pixels, over/underflow for this benchmark */
        for( x=1; x < XRES-1; x++ ) {
            i2 = (int)in[x+1];
            out[x] = (char)( (i0 + (2*i1) + i2) / 4 );
            i0 = i1; i1 = i2;
        }
        in += XRES;
        out += XRES;
    }
} /*filterF*/
```

- read a byte from one array, compute, store result in second array, a byte at a time.
- If the arrays line up on top of each other in a direct-mapped cache, there is massive cache-thrashing.

execution time via array size: [comp.arch posting]

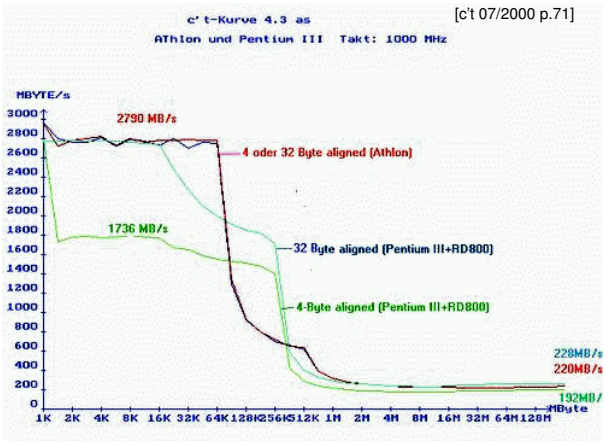
SYS	511	512	513	1023	1024	1025	2047	2048	2049	
CRIM	0.2	0.3	0.2	0.8	7.3*	0.9	3.7	33.4*	3.4	D
INDIGO4K	0.2	0.3	0.2	0.8	9.4*	0.8	3.2	37.9*	3.2	D
IN4K-fix	0.2	0.2	0.2	0.8	0.8	0.8	3.3	3.2	3.2	D
HP 720	0.3	0.7	0.3	1.1	2.7*	1.0	4.2	10.8*	4.2	D
HP 735	0.1	0.6*	0.1	0.6	2.7*	0.6	2.4	11.1*	2.6	D
HP 735	0.1	0.7*	0.1	0.6	2.7*	0.6	2.2	10.8*	2.2	D
Gwy486-66	0.3	0.3	0.3	1.3	1.4	1.3	5.5	5.5	5.5	SA?

x86: Pentium III Caches. . .

Cache or Buffer	Characteristics
L1 Instruction Cache[1]	- P6 family and Pentium® processors: 8 or 16 KBytes, 4-way set associative, 32-byte cache line size; 2-way set associative for earlier Pentium® processors. - Intel486™ processor: 8 or 16 KBytes, 4-way set associative, 16-byte cache line size, instruction and data cache combined.
L1 Data Cache[1]	- P6 family processors: 16 KBytes, 4-way set associative, 32-byte cache line size; 8 KBytes, 2-way set associative for earlier P6 family processors. - Pentium® processors: 16 KBytes, 4-way set associative, 32-byte cache line size; 8 KBytes, 2-way set associative for earlier Pentium® processors. - Intel486™ processor: (see L1 instruction cache).
L2 Unified Cache[2,3]	- P6 family processors: 128 KBytes, 256 KBytes, 512 KBytes, 1 MByte, or 2 MByte, 4-way set associative, 32-byte cache line size. - Pentium® processor: System specific, typically 256 or 512 KBytes, 4-way set associative, 32-byte cache line size. - Intel486™ processor: System specific.
Instruction TLB (4-KByte Pages)[1]	- P6 family processors: 32 entries, 4-way set associative. - Pentium® processor: 32 entries, 4-way set associative; fully set associative for Pentium® processors with MMX™ technology. - Intel486™ processor: 32 entries, 4-way set associative, instruction and data TLB combined.
Data TLB (4-KByte Pages)[1]	- Pentium® and P6 family processors: 64 entries, 4-way set associative; fully set associative for Pentium® processors with MMX™ technology. - Intel486™ processor: (see Instruction TLB).
Instruction TLB (Large Pages)	- P6 family processors: 2 entries, fully associative - Pentium® processor: Uses same TLB as used for 4-KByte pages. - Intel486™ processor: None (large pages not supported).
Data TLB (Large Pages)	- P6 family processors: 8 entries, 4-way set associative. - Pentium® processor: 8 entries, 4-way set associative; uses same TLB as used for 4-KByte pages in Pentium® processors with MMX™ technology. - Intel486™ processor: None (large pages not supported).
Write Buffer	- P6 family processors: 12 entries. - Pentium® processor: 2 buffers, 1 entry each (Pentium® processors with MMX™ technology have 4 buffers for 4 entries). - Intel486™ processor: 4 entries.

x86: ctkurve

[c't 07/2000 p.71]



- Messung der Cache-Transferrate vs. Blockgröße (random)
- Caches deutlich sichtbar: Pentium 16K/256K, Athlon 64K/512K

x86: Pentium III Cache-Modi

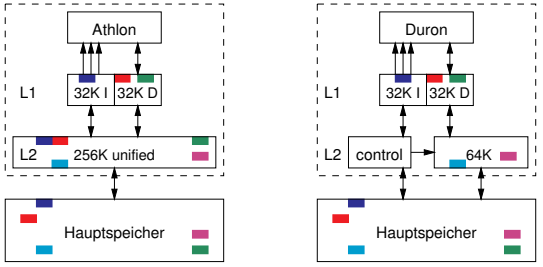
Caching Method	P6 Family Processors	Pentium® Processor	Intel486™ Processor
Uncacheable (UC)	Yes	Yes	Yes
Write Combining (WC)	Yes[1]	No	No
Write Through (WT)	Yes	Yes[2]	Yes[2]
Write Back (WB)	Yes	Yes[2]	No
Write Protected (WP)	Yes[1]	No	No

NOTES:

1. Requires programming of MTRRs to implement.
2. Speculative reads not supported.

- Write Combining (WC)—System memory locations are not cached (as with uncacheable memory) and coherency is not enforced by the processor's bus coherency protocol. Speculative reads are allowed. Writes may be delayed and combined in the write buffer to reduce memory accesses. The writes may be delayed until the next occurrence of a buffer or processor serialization event, e.g., CPUID execution, a read or write to uncached memory, interrupt occurrence, LOCKed instruction execution, etc. if the WC buffer is partially filled. This type of cache-control is appropriate for video frame buffers, where the order of writes is unimportant as long as the writes update memory so they can be seen on the graphics display. See Section 9.3.1., "Buffering of Write Combining Memory Locations", for more information about caching the WC memory type. The preferred method is to use the new SFENCE (store fence) instruction introduced in the Pentium® III processor. The SFENCE instruction ensures weakly ordered writes are written to memory in order, i.e., it serializes only the store operations.

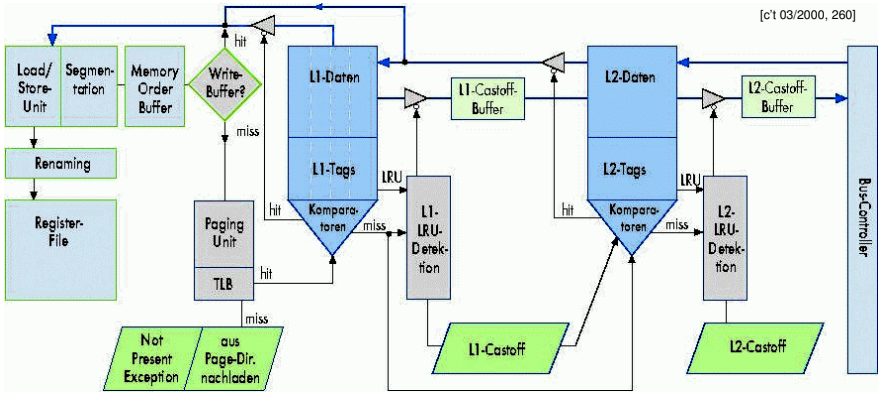
x86: AMD Duron: Cache



AMD Duron "exclusive" L2-Cache: => vgl. "victim buffer"

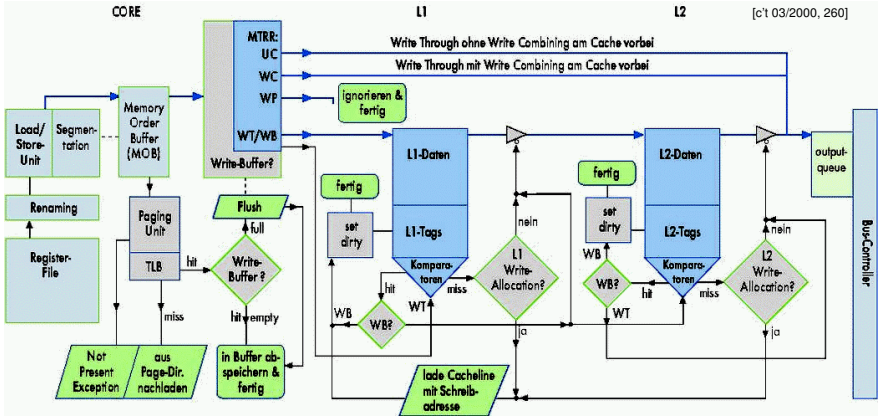
- L1-Cache: wie im Athlon (32KB + 32KB)
- L2-Cache: nur 64 KB statt 256 KB
- wäre bei herkömmlicher Verwaltung sinnlos (alle Daten doppelt)
- daher: L2-Cache speichert nur Daten, die nicht im L1 sind
- nur ca. 10% Performanceverlust

x86: Pentium II Lesezugriff . . .



- Cachezugriffe: L1 typ. 1..2 Takte, L2 typ. 2..10 Takte
- Speicherzugriffe: ca. 100 Takte

x86: Pentium II Schreibzugriff . . .



- MemoryTypeRangeRegister: schnelle I/O, z.B. Graphikkarte
- weitere Stufen (z.B. AGP GART) im Chipsatz ...

Speicherhierarchie: Übersicht, typ. Werte

	TLB	L1-Cache	L2-Cache	Virtueller Speicher
size / byte	32 .. 8K	1 .. 128K	256K .. 16M	16M .. 8G
block size / byte	4..8	4..32	32..256	4K..16K
hit time / clk	1	1..2	6..15	10..100
miss penalty / clk	10..30	8..66	30..200	700K..6M
miss rate / %	0.1- 2	0.5 .. 20	15 .. 30	0.000001 .. 0.001
backup	L1	L2	DRAM	Disks
block placement	FA	DM	DM / SA	FA
block identification	tags	tags	tags	table
block replacement	random	-	random	~ LRU
write strategy	flush	WT / WB	WB	WB

FA/SA/DM = full/set associative/direct mapped
WB/WT = write back/write through [H&P p.471]

Speicherhierarchie: Fazit

- performance gap wächst und wächst
- DRAM inhärent langsam

=> Speicherhierarchie wird immer wichtiger

- größere, tiefere Caches
- komplexere Caches: voll assoziativ, non-blocking, etc.
- aber Nutzen nur für "einfache" Anwendungen

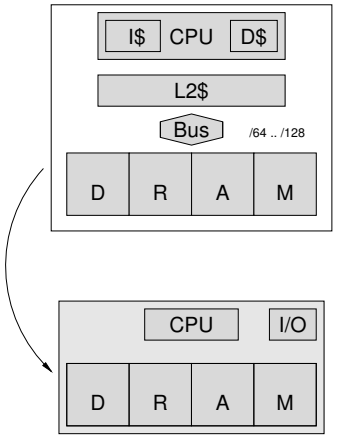
=> wichtige Forschungsaufgaben:

- intelligenteres Cache-Management
- Prefetching
- computational RAM / IRAM / . . .

IRAM: Konzept / "Vision"

IRAM := µP + DRAM + I/O auf einem Chip

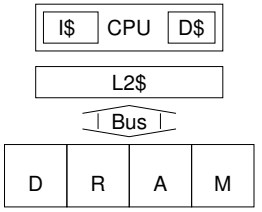
- Performance gap CPU/Speicher schließen:
  Latenz 5-10x 20ns statt 200ns
  Bandbreite 100x TB/s
- Speicherorganisation anpassen:
  beliebig wählbar: #bits, Busbreite, ...
- Energieverbrauch senken:
  kein DRAM-Bus: 2-4x
- Platzverbrauch senken:
  CPU passt auf DRAM: 2-4x



IRAM: performance gap "Tax"

Caches: kein Wert an sich, nur zum Schließen des performance gap

Beispiele:	%Fläche (~Kosten)	%Transistoren (~Leistung)
• Alpha 21164	37%	77%
• ARM SA110	61%	94%
• Pentium Pro	64%	88%



Patterson: performance gap "tax"

IRAM: Motivation

mehrfache Motivation:

- Stromverbrauch, Platzbedarf - insbesondere für mobile Geräte
- Anpassung von Speicherbedarf und -organisation
- Performance gap zwischen Prozessor und DRAM schließen,
  minimale Latenz, maximale On-chip Bandbreite
- neue Marktstrategie für DRAM-Produzenten (wegen Preisverfall)

Alternativen?

- neue, revolutionäre Chip-Packaging Technologien - unwahrscheinlich
- komplexere CPUs (out-of-order, multiple-issue, ...) - schwierig
- neue DRAM Standards (SDRAM, RAMBUS, ...) - nicht in Sicht

IRAM: Architektur?

1G Transistoren möglich, aber welche Rechnerarchitektur?

- ein Prozessor + DRAM:
  - Nutzen fraglich, evtl. langsamer als optimierte CPU + Cache + DRAM
  - verschenkt hohe on-chip Bandbreite, da #issues < 8
  - wenig innovativ
- SIMD oder MIMD Parallelrechner?
  - viele Prozessoren, aber nur wenig RAM / Prozessor
  -Programmierung ist ungelöstes Problem
  - alle bisherigen Varianten gescheitert
- Graphikprozessoren - bereits am Markt und etabliert
- I-VRAM := DRAM + RISC + Vektorrechner [Berkeley IRAM group]

IRAM: "vanilla" approach?!

vorhandenen Rechner (Alpha 21164) in DRAM Technologie implementieren

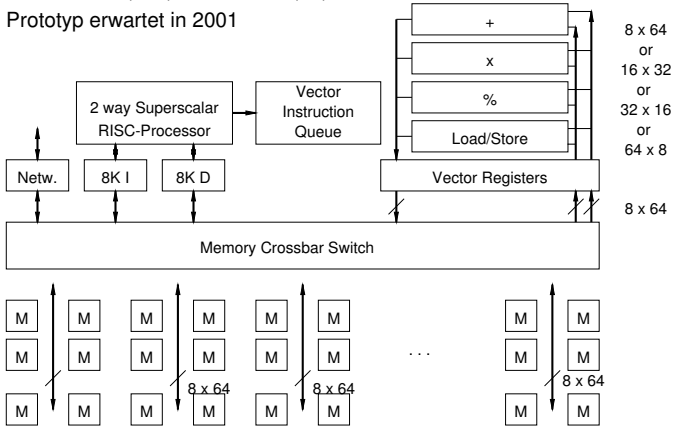
- gleiche Architektur: gleiche Caches, einfaches DRAM, ...
- übliche Benchmarks simulieren

Logik in DRAM Prozeß?	Faktoren: (optimistisch - pessimistisch)	
• Logik langsamer	1.3 - 2.0	
• SRAM (Caches) langsamer	1.1 - 1.3	
• DRAM schneller	10.0 - 5.0	
• SPEC92	0.8 - 0.6	langsamer!
• Database	1.1 - 0.9	gleich
• Sparse matrix	1.8 - 1.2	schneller

Performance nicht überzeugend, aber Leistung/Platzbedarf/Kosten besser

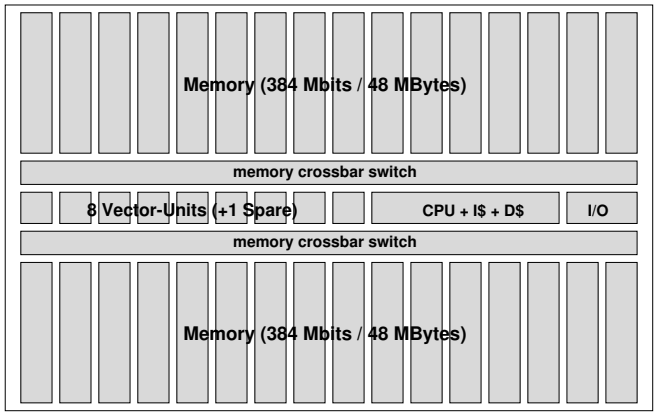
IRAM: V-IRAM 2

- 0.18 µm, fast logic, 1 GHz, 96 MByte DRAM
- 16 GFLOPS (64b), 128 GOPS (8b)
- Prototyp erwartet in 2001



IRAM: V-IRAM 2 Floorplan

- 0.18 µm, 1G Transistoren: 80% DRAM, 4% Vector, 3% CPU
- Größe und Redundanz wie 1Gb DRAM



IRAM: Zusammenfassung

Moore's Law: 1% / Woche

- Engpaß ist Performance gap zwischen CPU und DRAM
- radikal neue Speichertechnologien zunächst unwahrscheinlich
- Technologie ermöglicht CPU und DRAM auf einem Chip ab 1998/1999

IRAM Potential

- Bandbreite 100x, Latenz 5-10x, Leistung 2-4x
- V-IRAM als Technologiedemo? (Graphikchips bereits lieferbar!)
- V-IRAM: 25-100MB Speicher @ 20ns, 4-16 GFLOPS, serielle I/O
- V-IRAM: 1 TB/s Bandbreite, Smart-SIMMs = TFLOPS

dramatische Auswirkungen auf Halbleiter-Markt

- wer liefert DRAM, wer liefert Mikroprozessoren?