Now we'll really get our hands dirty, and try to become DRAM designers. That is, we want to understand the tradeoffs, and design our own memory system with DRAM cells. By doing this, we can gain some insight into some of the basis of claims by proponents of various DRAM memory systems.

A Memory System is a system that has many parts. It's a set of technologies and design decisions. All of the parts are inter-related, but for the sake of discussion, we'll split the components into ovals seen here, and try to examine each part of a DRAM system separately.

Memory System Design

- Clock Network
- I/O Technology
- Topology
- Chip Packaging
- DRAM Chip Architecture
- Pin Count
- Address Mapping
- Access Protocol
- Row Buffer Management
- DRAM Memory System
What is a “good” system? It’s all about the cost of a system. This is a multi-dimensional tradeoff problem. Especially tough when the relative cost factors of pins, die area, ... a while, but we’ve quickly progressed through EDO, SDRAM, DDR/RDRAM, and now DDR II and whatever else is on the horizon.

DRAM System Design

- Package Cost
- Interconnect Cost
- Logic Overhead
- Test and Implementation
- Power Consumption

Bandwidth

Latency
Now we’ll really get our hands dirty, and try to become DRAM designers. That is, we want to understand the tradeoffs, and design our own memory system with DRAM cells. By doing this, we can gain some insight into some of the basis of claims by proponents of various DRAM memory systems.

A Memory System is a system that has many parts. It’s a set of technologies and design decisions. All of the parts are inter-related, but for the sake of discussion, we’ll split the components into ovals seen here, and try to examine each part of a DRAM system separately.

- DRAM manufactures sell bits.
- Overriding consideration is bits per unit die area.
- Buffers, multiplexors, decoders, sophisticated control logic all have respective die cost.

**DRAM Evolution**

- Conventional DRAM
- FPM
- EDO
- BEDO
- SDRAM

Higher Performance
Higher Die Cost
Now we'll really get our hands dirty, and try to become DRAM designers. That is, we want to understand the tradeoffs, and design our own memory system with DRAM cells. By doing this, we can gain some insight into some of the basis of claims by proponents of various DRAM memory systems.

A Memory System is a system that has many parts. It's a set of technologies and design decisions. All of the parts are inter-related, but for the sake of discussion, we'll split the components into ovals seen here, and try to examine each part of a DRAM system separately.

* figure taken from SLDRAM white paper, by Pete Gillingham
Differences of Philosophy

**SDRAM - Variants**
- **Controller**
- **Complex Interconnect**
- **Inexpensive Interface**
- **Simple Logic**
- **DRAM Chips**

**RDRAM - Variants**
- **Controller**
- **Simplified Interconnect**
- **expensive Interface**
- **Complex Logic**
- **DRAM Chips**

Complexity Moved to DRAM
To begin with, we look in a crystal ball to look for trends that will cause changes or limit scalability in areas that we are interested in. ITRS = International Technology Roadmap for Semiconductors.

Transistor frequencies are supposed to nearly double every generation, and transistor budget (as indicated by Million Logic Transistors per cm\(^2\)) are projected to double. Interconnects between chips are a different story. Measured in cents/pin, pin cost decreases only slowly, and pin budget grows slowly each generation.

**Punchline:** In the future, Free Transistors and Costly Interconnects.

---

### Technology Roadmap (ITRS)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi Generation (nm)</td>
<td>90</td>
<td>65</td>
<td>45</td>
<td>32</td>
<td>22</td>
</tr>
<tr>
<td>CPU MHz</td>
<td>3990</td>
<td>6740</td>
<td>12000</td>
<td>19000</td>
<td>29000</td>
</tr>
<tr>
<td>MLogicTransistors/cm(^2)</td>
<td>77.2</td>
<td>154.3</td>
<td>309</td>
<td>617</td>
<td>1235</td>
</tr>
<tr>
<td>High Perf chip pin count</td>
<td>2263</td>
<td>3012</td>
<td>4009</td>
<td>5335</td>
<td>7100</td>
</tr>
<tr>
<td>High Performance chip cost (cents/pin)</td>
<td>1.88</td>
<td>1.61</td>
<td>1.68</td>
<td>1.44</td>
<td>1.22</td>
</tr>
<tr>
<td>Memory pin cost (cents/pin)</td>
<td>0.34 - 1.39</td>
<td>0.27 - 0.84</td>
<td>0.22 - 0.34</td>
<td>0.19 - 0.39</td>
<td>0.19 - 0.33</td>
</tr>
<tr>
<td>Memory pin count</td>
<td>48-160</td>
<td>48-160</td>
<td>62-208</td>
<td>81-270</td>
<td>105-351</td>
</tr>
</tbody>
</table>

**Trend:** Free Transistors & Costly Interconnects
So we have some choices to make. Integration of memory controller will move the memory controller on die, frequency will be much higher. Command-data path will only cross chip boundaries twice instead of 4 times. But interfacing with memory ... number of pins) AND lowest latency, we'll need custom RAM, might as well be SRAM, but it will be prohibatively expensive.
Instead of thinking about things on a strict latency-bandwidth perspective, it might be more helpful to think in terms of latency vs pin-transition efficiency perspective. The idea is that everything is bandwidth. Here are the different bandwidths:

- Clock
- Row Cmd/Addr Bandwidth
- Col. Cmd/Addr Bandwidth
- Write Data Bandwidth
- Read Data Bandwidth

Latency and Bandwidth

Pin-bandwidth and

Pin-transition *Efficiency (bits/cycle/sec)
Wave Pipelined Memory:

Unidirectional Topology:

- Write Packets sent on Command Bus
- Pins used for Command/Address/Data
- Further Increase of Logic on DRAM chips
Certain things simply does not make sense to do. Such as various STREAM components. Move multimegabyte arrays from DRAM to CPU, just to perform simple... the critical path of a DRAM access. That would serve to slow down a general access in terms of the “real latency” in ns.

Page Based Commands:

Instead of $A[\ ] = 0$; Do “write 0”

Why do $A[\ ] = B[\ ]$ in CPU?

Move Data inside of DRAM or between DRAMs.

Why do STREAMadd in CPU?

$A[\ ] = B[\ ] + C[\ ]$

Active Pages *(Chong et. al. ISCA ‘98)*
For a given physical address, there are a number of ways to map the bits of the physical address to generate the "memory address" in terms of device, row, column, and bank addresses. Initially, the cost of RDRAM was high due to the need for heat spreaders on the memory modules to prevent hotspots from building up.

Address Mapping:

- **Physical Address**
  - Device Id
  - Row Addr
  - Col Addr
  - Bank Id

Access Distribution for Temp Control
Avoid Bank Conflicts
Access Reordering for performance
Each Memory system consists of one or more memory chips, and most times, accesses to these chips can be pipelined. Each chip also has multitudes of banks, and most of the times, accesses to these banks can also be pipelined. (key to efficiency is to pipeline commands)

Access Reordering:

Multiple Banks to Reduce Access Conflicts

Read 05AE5700  Device id 3, Row id 266, Bank id 0
Read 023BB880  Device id 3, Row id 1BA, Bank id 0
Read 05AE5780  Device id 3, Row id 266, Bank id 0
Read 00CBA2C0  Device id 3, Row id 052, Bank id 1

More Banks per Chip == Performance == Logic Overhead
Access Reordering:

1. Read 05AE5700 → Device id 3, Row id 266, Bank id 0
2. Read 023BB880 → Device id 3, Row id 1BA, Bank id 0
3. Read 05AE5780 → Device id 3, Row id 266, Bank id 0
4. Read 00CBA2C0 → Device id 1, Row id 052, Bank id 1

Act = Activate Page (Data moved from DRAM cells to row buffer)
Read = Read Data (Data moved from row buffer to memory controller)
Prec = Precharge (close page/evict data in row buffer/sense amp)
Each Load command is translated to a row command and a column command. If two commands are mapped to the same bank, one must be completed before the other can start.

Or, if we can re-order the sequences, then the entire sequence can be completed faster.

By allowing Read 3 to bypass Read 2, we do not need to generate another row activation command. Read 4 may also bypass Read 2, since it operates on a different row. But I put in the precharge explicitly to show that two rows cannot be active within tRC (DRAM architecture) constraints.

Serial Link Memory:

*ref: Poulton ISSCC 1999 signaling tutorial
Each Load command is translated to a row command and a column command. If two commands are mapped to the same bank, one must be completed before the other can start. Or, if we can re-order the sequences, then the entire sequence can be completed faster.

By allowing Read 3 to bypass Read 2, we do not need to generate another row activation command. Read 4 may also bypass Read 2, since it operates on a different row. However, I put in the precharge explicitly to show that two rows cannot be active within tRC (DRAM architecture) constraints.
Each Load command is translated to a row command and a column command. If two commands are mapped to the same bank, one must be completed before the other can start. Or, if we can re-order the sequences, then the entire sequence can be completed faster.

By allowing Read 3 to bypass Read 2, we do not need to generate another row activation command. Read 4 may also bypass Read 2, since it operates on a different bank. However, the timing constraints of the DRAM architecture require that two rows cannot be active simultaneously. Therefore, I put in the precharge explicitly to show that two rows cannot be active within tRC constraints.

- Code in structural and behavioral Verilog
- Target (High Performance/Low power/Low cost)
- Interface with 16 bit RISC CPU
- Generic so it can be reused
Each Load command is translated to a row command and a column command. If two commands are mapped to the same bank, one must be completed before the other can start.

Or, if we can re-order the sequences, then the entire sequence can be completed faster. By allowing Read 3 to bypass Read 2, we do not need to generate another row activation command. Read 4 may also bypass Read 2, since it operates on a different row. Notice that the command sequence is pipelining from the memory controller to the system controller. But I put in the precharge explicitly to show that two rows cannot be active within tRC (DRAM architecture) constraints.

**Simulator**

- Code in C and C++
- Target High Performance
- Interface with Simplescalar or Bochs
- Generic so it can be reused