DRAM Reliability:
Parity, ECC, Chipkill, Scrubbing

Alpha Particle or Cosmic Ray

electron-hole pairs

silicon
Alpha Particles:

- Soft errors were big problems for early DRAM chips.
- Low energy alpha particles were discovered to be the culprit, but where were they coming from?
- Intel published paper in 1979 caused industry to pay close attention to material purity in silicon processing and packaging.
- Now largely considered as “solved problem”
Cosmic Rays:

- High energy cosmic rays originate in space.
- Number of cosmic ray events depend on altitude.
- IBM claims 5950 failures per billion device-hours at sea level, 0 failures in underground vault, with 50 feet of rocks completely shielding test setup.
- Computers are more susceptible to cosmic rays in Denver and on space station.
- Computers designed for space applications need to account for effects of cosmic ray
Parity: “For Farmers”

- Odd bit error detection
- No error correction capability
- Overhead: 1 bit per byte
Error Correcting Code I

- Also based on “parity checking”, but more sophisticated
- Error detection AND correction capability
- Overhead: depending on scheme
**Error Correcting Code IIa**

**Single-bit Error Correction (SEC)**

- Requires n+1 check bits to provide SEC to $2^n$ data bits

**Diagram:**

- Start with 8 data bits (do not use $D_0$)
- Reserve $R_m$ bit positions where $m$ is a power of 2.
- Move data bits into available bit positions. (skip $R_0$)
- Display “$m$” in binary format.

**Equations:**

$R_{0001} = R_{0011} + R_{0101} + R_{0111} + R_{1001} + R_{1011}$

$R_{0010} = R_{0011} + R_{0110} + R_{0111} + R_{1010} + R_{1011}$

$R_{0001} = R_{0011} + R_{0101} + R_{0111} + R_{1001} + R_{1011}$

- $R_{0010}$
Error Correcting Code IIb

SEC Encoding Example

- Start with 8 data bits (do not use D₀).
- Reserve Rₘ bit positions where m is a power of 2.
- Move data bits into available bit positions. (skip R₀)
- Display “m” in binary format.
- Rₘ bit positions will be the check bits, where each Rₘ bit will store the parity of the other bit positions where the mᵗʰ bit in the index is set.

\[
\begin{align*}
D &= \{11001110\} \\
R &= \{011010011110\}
\end{align*}
\]
Error Correcting Code IIc

SEC Verification Example

R = \{ 0 1 1 0 1 0 0 1 1 1 1 0 \}
R = \{ 0 1 1 0 1 0 0 1 1 0 0 \} One bit error. Can we detect and correct?

Recompute check bits

\[ R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1 \]
\[ R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0 \]
\[ R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 0 = 0 \]
\[ R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 0 \oplus 0 = 0 \]

XOR old check bits against new check bits

\[
\begin{array}{ccccc}
R_{1000} & R_{0100} & R_{0010} & R_{0001} & \\
1 & 0 & 1 & 0 & \\
+ & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & 1 & 1 & \\
\end{array}
\]
Old
New
Difference !

Bit position 11 is rotten
Error Correcting Code IIIa

What about multi-bit errors?

\[ R = \{ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \} \]

\[ R = \{ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \} \]  Multi bit error. Can we detect and correct?

Recompute check bits

\[ R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1 \]
\[ R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0 \]
\[ R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 1 = 1 \]
\[ R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 0 \oplus 1 = 1 \]

XOR old check bits against new check bits

\[
\begin{array}{cccc}
R_{1000} & R_{0100} & R_{0010} & R_{0001} \\
\oplus & 1 & 0 & 1 & 0 \\
\oplus & 1 & 1 & 0 & 1 \\
\hline
0 & 1 & 1 & 1
\end{array}
\]

Old

New

Difference !

Oops, Bit position 7 is NOT rotten
Error Correcting Code IIIb

What about multi-bit errors?

Single Error Correction Double Error Detection (SECDED)

- requires $n+2$ check bits to provide SECDED to $2^n$ data bits
Error Correcting Code IIIc

What about multi-bit errors - Redux

R = \{ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \}\n
Multi bit error. Can we detect and correct?

Recompute check bits

R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1

R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0

R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 1 = 1

R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 0 \oplus 1 = 1

XOR old check bits against new check bits

<table>
<thead>
<tr>
<th>( R_{1000} )</th>
<th>( R_{0100} )</th>
<th>( R_{0010} )</th>
<th>( R_{0001} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( \oplus )</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

| 0 | 1 | 1 | 1 | Difference ! |

XOR check bits tell us there is error, but \( R_0 \) parity says all is well. This is a 2 bit error, cannot be corrected.
Error Correcting Code IV

- SECDED requires $n + 2$ check bits to protect $2^n$ data bits
- Data bus width of $64 = 2^6$ requires $6 + 2 = 8$ check bits to provide SECDED protection
- Logic depth of $n + 1 = 7$ to compute XOR parity for $0^{th}$ bit
- May cost additional cycle(s) on read latency
Weaknesses of ECC?

What if this chip dies? (hard failure)

Future low power, smaller cell, smaller capacitance DRAM may be more susceptible to

Alpha Particle or Cosmic Ray

Does not work with masked (partial) writes to DRAM

Multiple bits from same chip may be corrupt

wide (per chip) data bus is not good for fault tolerance

Error rate is given in failures per bit. There are always more DRAM storage bits in the next generation system.
“Chipkill” I

Separate ECC protected words, each can detect and correct single bit error

Architect the memory system so there is no Single Point of Failure that could bring down the system
“Chipkill” II

SECDED requires \( n + 2 \) bits to protect \( 2^n \) bits. Need 9 check bits to protect 128 data bits.

Deploy more advanced algorithm to detect and repair multi-bit errors with 128 data bits and 16 check bits, or 256:32.

Architect the memory system so there is no Single Point of Failure that could bring down the system. Deploy method 1, method 2, or combination of both to protect against multi-bit errors.
Problems Remain

ECC/Chipkill protects data transmission error in addition to Single Event Upset storage error.

Address/command transmission errors are not protected.

DQS is per byte for the x8 and x16 chip. “topology matched, source synchronous” is a problem for certain types of chipkill implementation.

DDR SDRAM
Scrubbing

Soft error model based on Single Event Upset alpha particles or cosmic rays.

“Scrubbing” merely reads out data to controller, scrub out any correctable error(s), write it back into memory before multi-bit errors build up and become no longer correctable.
Serverworks Grand Champion HE

- 128 bit ECC algorithm. 16 bit detection, 8 bit correction.
- Memory scrubbing
- Spare memory
- Memory mirroring
- Hot plug memory card
What about Rambus?

Each “access” to DRAM is serviced by a single DRAM chip. One DRAM chip will provide 8 consecutive beats of data, 16 bit wide per beat.

- Design ECC version, with 18 bit wide interface. provides SECDED protection, not chipkill
Interleaved Device Mode

- Each chip provides 2 bits of data for every read request
- Provides effective chipkill capability when used in multiple channel configuration (i.e. EV7)