DRAM Reliability:

Parity, ECC, Chipkill, Scrubbing

- High energy terrestrial neutron
- High energy alpha particle
- 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 to be “solved problem”
Terrestrial Neutrons:

Space aliens bent on destroying human technology

- High energy cosmic rays originate in space, but ...
- collisions with atmosphere generates secondary particles. “Terrestrial Neutrons” main part of flux
- Flux of neutrons 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.
Parity: “For Farmers”

- Odd bit error detection
- No error correction capability
- Overhead: 1 bit per byte

Even or odd parity
Parity check

Memory Controller

DRAM

x1 chip
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.

- \( R_0001 = R_{0011} \lor R_{0111} \lor R_{1011} \)
- \( R_{0010} = R_{0011} \lor R_{0111} \lor R_{1011} \)

\( R_m \) bit positions will be the check bits, where each \( R_m \) bit will store the parity of the other bit positions where the \( m^{th} \) bit in the index is set.
Error Correcting Code IIb

SEC Encoding Example

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

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

D = \{1 1 0 0 1 1 1 0\} → R = \{0 1 1 0 1 0 0 1 1 1 0\}
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 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} + 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} + 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

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

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 \}  \quad \text{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} \\
1 & 0 & 1 & 0 \quad \text{Old} \\
\oplus & 1 & 1 & 0 \quad \text{New} \\
\hline
0 & 1 & 1 & 1 \quad \text{Difference !}
\end{array}

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

Start with 8 data bits

Basic Idea: Use $R_0$ to check parity of data bit vector, (data bits only)

$$R_0 = D_1 \oplus D_2 \oplus \ldots \oplus D_2^n$$
Error Correcting Code IIIc

What about multi-bit errors - Redux

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

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

Multi bit error. Can we detect and correct?

Recompute check bits

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

\[ R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1101} = 1\oplus 0\oplus 0\oplus 1\oplus 0 = 0 \]

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

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

XOR old check bits against new check bits

\[ \begin{array}{ccccc}
R_{1000} & R_{0100} & R_{0010} & R_{0001} \\
1 & 0 & 1 & 0 & \text{Old} \\
\oplus & & & & \text{New}
\end{array} \]

\[ \begin{array}{ccccc}
& 1 & 1 & 0 & 1 \\
\hline
0 & 1 & 1 & 1 & \text{Difference !}
\end{array} \]

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.
- SECDED needs \( n + 2 \) check bits to protect \( 2^n \) data bits
- Data bus width of \( 64 = 2^6 \) means \( 6 + 2 = 8 \) check bits to provide SECDED protection
- Logic depth of \( n + 1 = 7 \) to compute XOR parity for 0th 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 high energy alpha particle or neutron

does not work with masked (partial) writes to DRAM

multiple bits from same chip may be corrupt

Error rate is given in failures per bit. There are always more DRAM storage bits in the next generation system.
Multi-bit Error Correction I

\[ 0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad 1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \alpha = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \alpha^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \]

Parity check matrix in \( \text{GF}(2^2) \)

\[
C_{\alpha^2} = \begin{bmatrix} 1 & 1 & 1 & \alpha & \alpha^2 & 1 & 1 & 1 & \alpha & \alpha^2 & 1 \\ 0 & 0 & 1 & 1 & 1 & \alpha & \alpha^2 & 1 & 0 & 0 & 0 \end{bmatrix}
\]

\[
C_{\alpha} = \begin{bmatrix} \alpha & 1 & 0 & 0 & 0 & 1 & 1 & 1 & \alpha & \alpha^2 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & \alpha & \alpha^2 & 1 & 0 & 0 & 0 \end{bmatrix}
\]

Apply transform matrices

\[
T_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad T_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad T_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad T_0^2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}
\]

Parity check matrix in binary field
Multi-bit Error Correction II

- Each pair of bit positions treated as a single symbol.
- Combine with bit steering to cover failure across address boundaries.
- Different algorithms exist with varying level of complexity
- Should try to work with established framework of (64, 72) DIMMs.
- Else, custom memory modules for specialized systems
“Chipkill” 1

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

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

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

Address/command transmission errors are not protected.

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